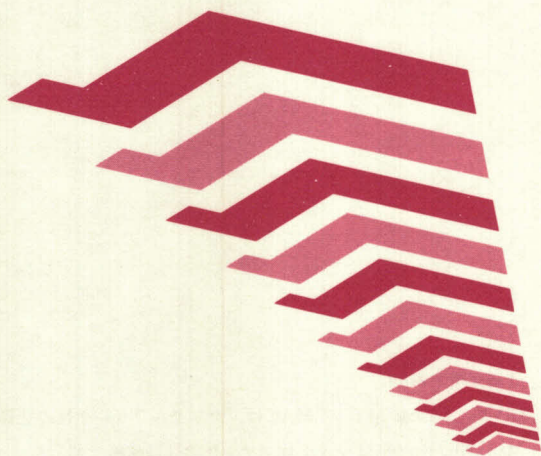
実戦マクロ・アセンブラ活用法

プロの要求を満たすMACRO 80のすべて

中野正次 著

CQ出版社

実戦マクロ・アセンブラ活用法

プロの要求を満たすMACRO 80のすべて

中野正次 著

CQ出版社

まえがき

① LET　ANS　:=　NUM1　*　NUM2

② .　　ABC+DEF−G : H

上の2つの表現は共にマクロ・アセンブラの応用例です．①はNUM1とNUM2を乗算してANSに入れる動作を，②は論理演算でA〜Gの各1ビットを

$(A \cdot B \cdot C + D \cdot E \cdot F) \cdot \overline{G}$

のように計算してHに入れる動作を表しています．アセンブラでもこのような表記ができることを御存じでしたか？

Basic や Pascal などの高級言語が普及している今日ですが，制御関係や高速処理が必要な分野ではまだまだアセンブラに頼らざるを得ないのが現状です．しかし，このアセンブラというものは生産性が低く，何とかしなければと頭を悩ませている担当者も多いのではないかと思います．制御関係でもすべてが高級言語化できないわけではありませんが，入出力管理をすべて組み込まなければならない OS のないシステムや，多機能のリアルタイム並行処理，そして速度が絶対優先の画像処理など，一般の高級言語では扱いきれない分野が少なくありません．

アセンブラの問題点は，記述性，読解性，デバッグ性，保守性などどれを取っても良いとはいえず，結果として見積もりと実績が一致しにくいので大幅な赤字を生むこともしばしば発生します．

マクロ・アセンブラは，マクロ機能のないアセンブラ（機械語アセンブラ）と高級言語の間を埋める性格を持っており，工夫次第では記述性〜保守性とも大幅な向上が図れます．さらに，必要があればいつでも全ステップにわたって機械語レベルで管理できるという柔軟性をも兼ね備えているので，使用するに当たっても不安な部分はほとんどありません．

マクロ・アセンブラは，CP/M80 の下では M80（Macro-80 Microsoft 製）が代表的で，Intel と Zilog の両ニモニックを処理できるので，実際に多く使用されています．けれどもその割にはマクロ機能を有効に利用している例は少ないようです．それというのも，機械語や制御用プログラムの解説書がたくさん出ている中に，マクロ機能について詳しく解説している本が少ないからでしょう．また，その少ないマクロの解説書がすべて MAC を基本にしていて，主として OS の下で動作するプログラムに使用している例を扱っており，OS の有効利用を目的にしています．これは，CP/M が MAC を使用して書かれているこ

との必然的結果ですが，これに対して組み込み用プログラムに焦点を定めたマクロ・アセンブラの解説書が待ち望まれていました．

本書はこの要望に応えるべく，ROM 化される前提のプログラムに対してのマクロ・アセンブラの利用法を解説するものです．また，我が国では，とりわけ組み込み用ソフトを扱う分野ではZ80が主流であると考えてM80のZilogフォーマットで使用例を示しています．というわけで，唯一ともいえるZ80用マクロ・アセンブラ解説書である本書の内容は入門レベルから高度な応用まで含めた構成になっています．

マクロ・アセンブラが真価を発揮するためには，何ステップかの機械語のグループをマクロ命令に置き換えるという単純な機能だけでは不十分で，これに条件アセンブル，数式処理，アセンブル変数，リロケータブル・モジュール，高級言語のオブジェクトとのリンク，リスト・フォーマットの変換など各ソフトウェアの機能を有機的に組み合わせて練り上げていく必要があります．

本書の前半は，通常のマクロ機能を利用したビット操作や領域確保，高精度四則，入出力ICのイニシャライズ，レジスタ群の **PUSH, POP,** CP/Mとのインターフェースなどを解説します．後半ではマクロの高度な応用としてマクロを1度だけ展開させる手法，他のプロセッサ用のクロス・アセンブラ，クロス・コンパイラ，四則(八則)のコンパイラ風表記，単一タイマによるマルチソフト・タイマ，ロジック回路シミュレータなどについて解説し，さらにコンパイラ（Pascal/MT＋）とのリンク，コンパイラでリスト変換を行ってM80でカナ文字コメントやカナ文字メッセージを処理する方法も紹介します（リスト中にカナ文字が入っているのは，すべてこの方法で処理したものです）．

マクロ・アセンブラを駆使するには，機械語に十分慣れている必要があります．本書においては機械語はすでに理解されていることを前提にしています．

マクロ機能は本質的にはハードウェアに依存しません．すなわち，CPU が変わっても，マクロ命令だけを使って書かれたプログラムは全く変更せずに使用できる可能性があります．マクロ定義の部分だけを変更すればよいのです．このことは多くの似たようなアプリケーションを小量生産する場合に重要なことです．もちろん，コンパイラも有効です．この点を考慮して，マクロ，そしてコンパイラを使用すれば，Z80CPU に全く固執することなく，現在開発しているソフトウェアが将来にも生き続けて行けるのです．機械語のみではCPUの変化に耐えられないことは明白です．

本書を手引き，足がかりとして，心血注いで積み上げたソフトウェア資源を有効利用するのに役立つことを心から祈ります．

1985年　著　者

□目　　次□

第1章
マクロ・アセンブラとは

マクロ・アセンブラはその機能の面から最も簡単にいえば,「アセンブラの短縮ダイヤル」のようなものです．短縮ダイヤルは憶えやすく，間違いが減り，ダイヤル時間を節約でき，指のエネルギ消耗も少なくてすむ(?)などのメリットがあります．また，短縮ダイヤルを憶え違いしたとしても全くの他人につながる心配がありません．これらの特長は，そのままマクロ・アセンブラのマクロ機能についてもあてはまります．

しかし，便利な短縮ダイヤルも，その存在を知らなかったり，使用法を知らなければ何も役には立ちません．また，使用法を知っていても，あらかじめ本当の番号を記憶させる作業を行わなければやはり実際に使用することはできません．このあたりの手続きもマクロ・アセンブラと非常によく似ています．

言葉の上では，マクロというのは巨視的とか大局的な意味を持っていて，細部を意識しないで使えるアセンブラとも考えられます．もう少しわかりやすく表現すれば「代表命令アセンブラ」とでもいうところでしょうか．そして，何をどのように代表させるかは，すべて短縮ダイヤルと同様にユーザが自由に決められるのです．このとき，代表の名前をつけておいて，使用したいときに代表名を書けばその中身をすべて書いたことと同じ効果になるというわけです．

自由に決められるとはいっても，やはりアセンブラですから，使える文字の種類や文字数を越えて使うことは無理です．それでも，機械語命令よりははるかに自由な表現が可能です．

```
.
A
LOAD    P, Q, R
???
ENZAN    Y = X + Z
```

```
ASOBI
LOGIC    AA  BB  +  CC  ¥  DD
```

など，すべてしかるべき定義をすれば代表命令すなわちマクロ命令として使用可能です（¥はASCIIコードでは＼です．本書ではカナ文字使用の都合上すべてJISコードで表示します）．これを見ると機械語とはずい分ようすがちがいます．マクロ・アセンブラ初体験の人には“奇怪語？！”かも知れません．マクロ機能を使用している人でも，通常のマクロ命令とはいささか趣が違うとお感じになると思います．このようにマクロ・アセンブラというものはかなり奥の深いものです．

本章では技術的内容に入る前の予備知識としての概要と他のソフトウェアとの関連，有効な利用法について述べます．

1.1 マクロ・アセンブラの効用

マクロ・アセンブラを使えば手間が省ける……といってしまえばそれまでですが，これを単にキーボードのキーを押す回数が減るから手間が省けると考えるなら大きな間違いです．マクロ・アセンブラを使用するメリットは，

①ソフトウェア資源の共通利用
②読解性，保守性の向上
③コーディング・ミスの防止
④分野別，目的別の柔軟な対応
⑤機密性
⑥（付随的に）一貫性，系統性の向上

などで，これらのどれを見てもプログラミング作業の中から単純作業の部分を削除する効果が期待できるものです．

ソフトウェア資源の共通利用については，サブルーチンの共用の形でマクロ・アセンブラでなくても可能ですが，マクロ使用では表現形式の自由度が大きいので，共用できる範囲が大幅に拡大します．

読解性のよさも表現形式の上にプログラムの意図を反映しやすいことによります．読解性がよければ担当者が変わったり，客先へ引き継いだりする場合にも少ない手続きで意思疎通ができ，誤解を招く確率も減少します．この結果，当然保守性も向上するわけです．

機械語では他人が書いたものはもとより，本人が書いたプログラムですら時間が経つと何をやりたいのか読み取るのに苦労しますが，これもマクロ表記によってはるかに思い出しやすくなります．

コーディング・ミスというのはフローチャートが正しいのに，コーディングする段階で発生するミスのことです．機械語では打鍵回数が多いので当然キー・ミスも増えがちですが，その他に **PUSH** と **POP** の順序を間違えたり **LD B,（DE）** と書いてしまったり，**IN A, PORT** とカッコを忘れたり，**SUB A, 2** と書くなどの単純なミスがしばしば起こります．デバッグ済みのマクロを使用するときには，この種のミスは発生しません．

マクロ・アセンブラはサブルーチン方式よりもはるかに柔軟な対応性があります．また，市販のコンパイラとちがって偏見に満ちた使い方もできます．マクロ命令はユーザ定義(マクロ・アセンブラの種類によっては，すでにあらかじめ組み込まれたマクロ定義を内蔵しているものもあります．M80 には組み込みマクロ定義はありません）ですから，ユーザの都合によって使用する分野や，作業目的に応じて定義することができます．このとき，共通に利用する範囲以外での有用性を考える必要は全くありません．高速処理か，高精度か，省メモリかによって必要最小限のマクロ定義をしておけばよいのです．逆に，メイン・プログラムが完成して正常に動作した後で，スピードを少し上げたいという場合，マクロ定義だけを変更して対応することも可能です．これは保守性の面でも有利になります．

機密性については読解性と相反する条件になりますが，ここでは 2 種類の意味が含まれています．ひとつは，メイン・プログラムのリストを公開してもマクロ定義を公開しなければ簡単にコピーされてしまうことはないということ．そして，特に機密保持用のマクロ定義を作ることもできるという 2 点です．

以上のようなメリットを最大限に発揮させるためには，ただ闇雲にマクロ定義を増やしても効果はそれほど上がりません．目的とするシステムで頻繁に使用される演算や手続きを把握し，課内や社内で共通利用できるように，また将来にわたっても仕様変更に耐えるように総合的な見通しをたてることが，マクロ・アセンブラを有効利用する上では不可欠な要素になります．このことを意識すればマクロの利用以外についても，ソフトウェア全体について系統性がよくなり，ムダな部分は目につくようになって自然に減って行きます．

以上が一般的な使い方においてのマクロ・アセンブラの効用ですが，特殊な用途としてすべてマクロ命令で表記したプログラムは他の CPU に簡単に移植できることから，複数機種の CPU で似たようなソフトウェアを開発するのに利用できます．また，別の CPU の機械語を生成するクロス・アセンブラとして利用する方法もあります．さらには，インタ

プリタ用の中間言語を生成したり，ハードウェアを改造して外部に命令解釈回路を付加した時に，マクロ定義でその命令も他の機械語と同等に扱えます．

マクロ・アセンブラの利用技術は，M80に固有のものではありません．もちろん，Z80や8080専用のものでもありません．各CPUの機械語に慣れることは，もちろんそれなりに作業効率の向上につながりますが，あくまでそのCPUに限定されていることに注意しなければなりません．これに対して，マクロ機能の利用技術はCPUが16ビットになろうと32ビットになろうと，基本的には共通した手法なのです．ここに長い目で見た時のマクロ利用の大きな効用が隠されています．

これらの他にもマクロ利用のメリットは細かく挙げればきりがありません．第2章以降を通読すればこの意味がよく理解できることと思います．

1.2　純機械語とアセンブラ

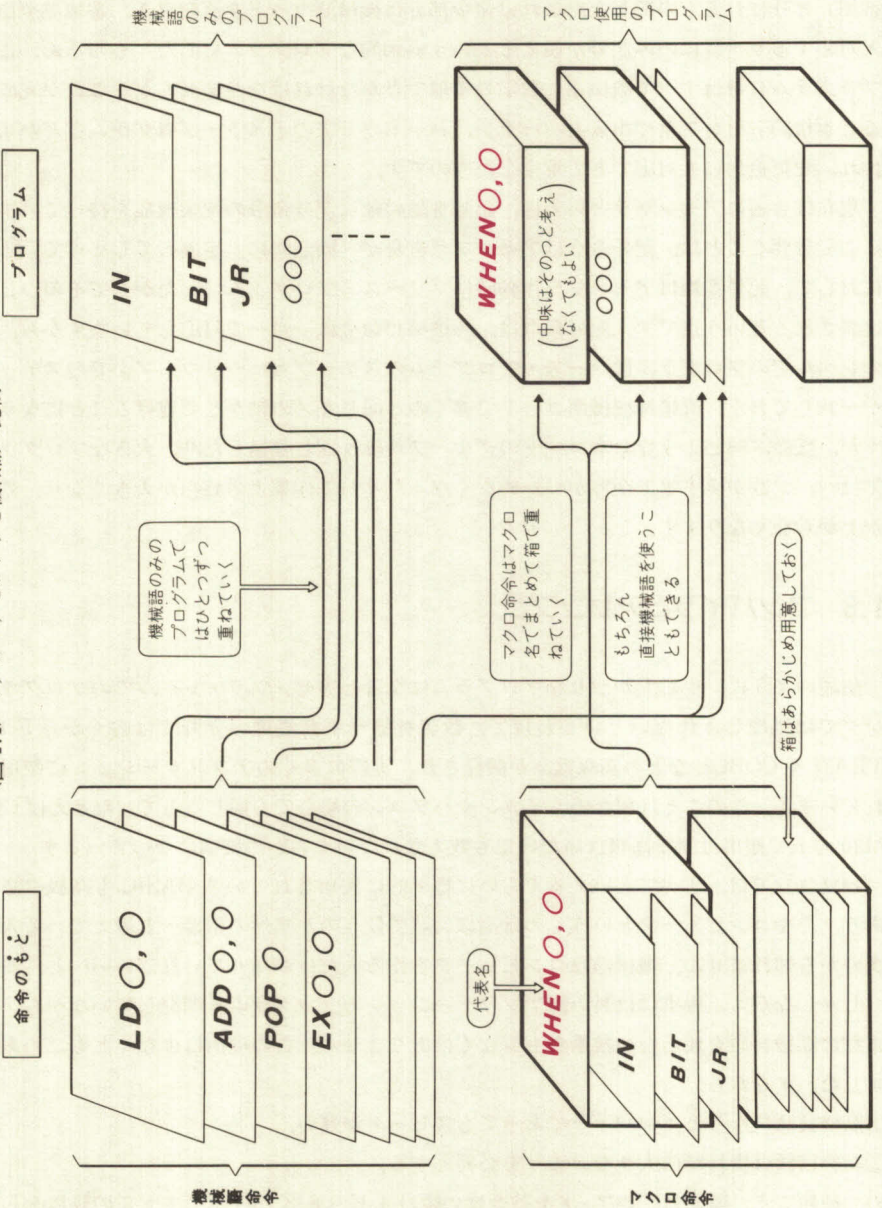
純粋に機械語といえば“0”と“1”だけのバイナリ・コードです．これでは人間が認識したり，表現したりするのに不都合すぎるということで，通常は8進法や16進法を使って表現します．パソコンでBASICから呼び出して使用する機械語プログラムはこのレベルで使用されています．すなわち，16進表記によって書かれているのです．それでは，その機械語プログラムを最初に作った人は16進数で考えたのでしょうか？　おそらく，16進に置き換える前にアセンブラで考えたと思います．実際，BASICの中で機械語を使用している場合に，アセンブラのソース・プログラムないし，リストが示されているものもあります．

もっとも，ある程度慣れてくれば，いきなり，

3E　FF　32　40　FF　DB　08　CB　6F　28　FA……

のように直接機械語で考えることも不可能ではありません．ところが，少し大きなプログラムになると絶対に困る部分が出てきます．それは，**C3**や**CD**（**JP**や**CALL**）などのプログラムの中を参照する絶対番地の命令です．これらの命令は最初にプログラムを作成する段階でやっかいなだけでなく，プログラム完成後でも，変更したり追加したりすると書き換えなければならなくなります．**3E**や**C3**などの命令コードは記号化されていなくても暗記すれば使えますが，飛び先番地だけは暗記しようがありません．

筆者がミニコンのハードウェアを設計していたころ，最初に作るソフトウェアは「記号

〈図1.1〉マクロ・アセンブラと機械語の比較

番地」と呼ばれるものでした．これは命令部分は機械語コードを直接書き，番地参照部分と対応するラベルにのみ記号が使えるという原始的なプログラムでした．もちろん，このプログラム自身はすべて機械語と数字の番地で書かなければなりませんが，これが完成すると本格的アセンブラや出入力ルーチン，ハードウェアのテスト・プログラムなどが楽に作れ，変更追加にも対応できて重宝したものです．

現在は普通にアセンブラといえば，記号番地の他に記号命令の変換機能を持っています．ここで重要なことは，記号命令はアセンブラ自身が「固定的に」定義してしまっているのに対して，記号番地はアセンブラ作動時に「ソース・プログラムにしたがってそのつど」定義されるという点です．記号命令は，各機械語命令に一対一で対応していますから，このレベルでのプログラムはソース・プログラムのステップとハードウェア上でのステップが一致しており，直接純機械語コードで書くのと同じキメの細かさで書けることになりますが，反面，同じような命令コードのグループが繰り返し登場したり，大きなプログラムではページ数が多すぎて全容がつかめなくなったりして作業上の負担が大きくなり，効率が上がらなくなります．

1.3 コンパイラとアセンブラ

前述のように，少し大かがりなプログラムになるとアセンブラ・レベルでのプログラミングでは処理しきれないというわけで，数値計算や事務処理の分野では古くからFORTRANやCOBOLなどの高級言語が開発され，実際に多くのアプリケーションに使用されています．このことは現在のマイコンやパソコンの場合でも同じことで，たとえばCP/M80の上で使用可能な高級言語だけでも数えきれないくらいの種類になっています．

パソコンでは，すべてといえるくらいに標準的に使用されているBASICも高級言語であり，今やコンピュータというものは高級言語で使うのが当然の常識…と考えている人が多いかも知れません．機械語はコンピュータを作る人だけが知っていればいい…と．

しかしながら，現実には長い間アプリケーションのプログラムを開発していながら，いまだに高級言語を使わない技術者も少なくはありません．この理由は少なくとも二つありました．すなわち，

①高級言語になるとコンパイラであってもスピードが遅い．

②高級言語は実行時に大きなメモリを必要とする．

の2種類です．確かに，コア・メモリ全盛の頃は4 K～8 Kバイトですべての処理をしな

ければならず，②のメモリを喰う点は致命的でした．しかし，メモリ・コストが低下している今日，この点は実質的には大きな要素ではなくなっています．本書で使用(第6章)するコンパイラ Pascal MT＋も簡単なプログラムは8 Kバイト以内に納まります．これはPROMでいえば2764が1個です．他に作業用，スタック用のRAMが必要ですが，決して大がかりになるわけではありません．

スピードについてはどうでしょうか．高級言語でもパソコンに内蔵されているインタプリタ方式は確かに遅いといえますが，コンパイラ方式のものでは多くのアプリケーションに対して実用になるスピードが得られます．にもかかわらず，コンパイラを使うことに踏み切れない理由は何でしょうか．それはコンパイラの実行スピードには保証がないということでしょう．アセンブラでなら，機械語のステップで書いているわけですから，当然実行時間もあらかじめ計算できます．ところが，コンパイラでは簡単なプログラムであってもやってみなければわからない要素が多いので安心して使えません．

生産量の少ないものでは，ソフトウェアの原価が全体の価格に占める割合が大きいので，コンパイラでやってみてダメだったからアセンブラで書き直すなどという遠回りな作業をやっている余裕がありません．ダメかも知れないものなら最初からアセンブラで書いた方が確実だというわけで，最後までコンパイラには手を出さないのも止むを得ないといえます．

コンパイラにも，スピードの目安になるベンチマーク・テストが行われてそのデータが出てはいますが，そのほとんどがフーリエ変換やソートなど，科学計算や事務用の分野での性能にかかわるものです．特に，コンパイラでROM化するプログラムを作って，入出力処理やインタラプト処理の応答時間などを測定したデータは皆無に近いのではないでしょうか．

コンパイラが不得意とする機能は，

①ビット単位の入出力

②時間管理の厳しいもの……（通信制御など）

③高速応答……（サーボ制御など）

などの処理です．これらに関しては，仕様書を見ても不可能ではないといえるだけで，時間的なものが定量的に出せることはまずありません．正直なところ全く見当がつかないものです．

それでは，汎用言語のコンパイラがなぜ特定の分野に対して無力なのでしょうか？その理由は，多分コンパイラを作る人が上記のような分野に関与していないからでしょう．コ

ンピュータがこのような分野に利用されるようになってから日が浅いということもその理由かも知れません．とにかく，上記の分野がコンパイラというビルの間の谷間のように取り残されてしまっているのは事実です．

しかし，それだけではありません．コンパイラで作成するプログラムは，ほとんどの場合OSの管理下で実行されるのが前提となっており，OSなしでは実行不可能か，実行できても多くの機能に制限を受けるケースが多いのです．また，OSなしで実行するための手続きも詳しく説明されていなかったりもします．OSの下で実行するプログラムは，入出力はすべてOSまかせなので，スピードはOSに聞いてくれということになって，コンパイラ側では責任が持てないわけです．

一方，コンピュータ言語の一般的な説明書は数多く出版されていますが，これらもほとんどがOSを前提として書かれています．入出力の手続きはハードウェアに依存するものなので，ハードウェアに依存する部分は本来の言語機能とは関係ない部分として扱うことになり，OSを前提とせざるを得ないのも実際上しかたがないことといえます．

そして，システムを作る側では（特にBASICを組み込んだパソコンでは）ハードウェアについて詳細を説明せず，言語の側からは個々のシステムについては扱いきれないということになって，どうしても不明確な部分が残ってしまいます．これが，前述の三点に代表される谷間にあたります．

アセンブラでなら，I/Oポートとメモリ配置さえわかっていればどんなシステムに対してもプログラミングが可能なので，そういう機能を組み込んだコンパイラが出現すればよいのですが，前述のような背景を考えれば当分の間期待できそうにないといえそうです．

コンパイラにおいて不都合な面は演算精度やデータ長についても現れます．アセンブラで書けば，その都度必要最少限の精度で演算できますが，コンパイラではそうは行きません．たとえば，24ビットの整数を扱うことは，通常のコンパイラではできません．整数は16ビットか32ビット（倍精度）に限定されます．ところが，工業的な分野では16ビットでは不足で24ビットなら十分足りるという数値が多いのです．これを32ビットで扱えば当然スピードが落ちます．Z80ではAとHLだけを使って演算すれば高速にできますが，これが24ビットまでですから32ビットになるとスピードが大幅に下がってしまいます．

この他，インタラプト処理用に裏レジスタ（AF'，BC'……のこと）を空けておくか，処理スピードを上げるために全レジスタを駆使するかなどについても，アセンブラでならプログラマの意図通りになりますが，コンパイラで裏レジスタ使用可否のスイッチがついているものは見かけません．もし，このスイッチがついたコンパイラができても，裏レジスタの

うち，BC'とHL'だけは使ってもよいとか，IYとDE'を使ってはいけないなどの細かい指定までは受け入れられません．これをコンパイラで実現することは不可能といってよいでしょう．

インタラプト処理のためにレジスタを空けておきたい理由は，レジスタの内容を**PUSH**する必要がないだけでなく，データをレジスタに置いたままにできることです．通常の簡単な処理ではインタラプト用にレジスタを専有する場合とメインと共有する場合とで処理時間が2～2.5倍違ってきます．

以上のような細かい要求をすべて受け入れられる夢のコンパイラが出現するまで，どうしてもアセンブラでなければ書けないプログラムが存在し続けることになります．

1.4 アセンブラとコンパイラの長所を兼備するマクロ・アセンブラ

コンパイラに谷間ともいえる弱点が存在し続ける限り，どうしてもアセンブラ・レベルでのプログラミングを効率よく行う手段を考えなければなりません．これには，その対象としている分野なり業種なりに対応したサブルーチンやデータ構造の共通化が必要になって来ます．ただし，ここでいう共通は，他の分野との共通化を意味するものではなく，同一分野内での共通化を指しています．

データ構造や処理手順の共通化ができれば，ソフトウェアの共用が可能になります．この作業は機械語アセンブラだけでもできたわけですが，マクロ・アセンブラのマクロ機能を活かせば共通利用するための作業量を減らし，より少ない仕様書でより見やすく書きやすい形式に仕上げることができます．また，内容としては必要最小限の機能として開発時間を短縮できます．ちなみに，

```
LET ANS = NUM1 * NUM2
```

という表現形式をとれば，意図することは明白で説明の必要はありません．しかし，これだけならコンパイラと全く同じであり，めんどうなマクロ定義をしてアセンブラで書く意味がありません．これをマクロ・アセンブラで書いた場合は，内容的にコンパイラよりも自由に設定できるという大きなメリットが出せるのです．たとえば，上の式で各変数名がすべて24ビットの整数を表しているとすれば，コンパイラでは処理できません．さらにマクロ・アセンブラでなら，**ANS**と**NUM1**は24ビットで**NUM2**は16ビットという構成もとれます．これで精度が足りれば，各数値が同一精度でなければならない理由は特にな

く，精度を下げた分だけスピードが上がります．

また，**ANS** と **NUM1** は符号付きで，**NUM2** は符号なしという組み合わせもできます．**NUM1** が位置を表し，**NUM2** は測定系の補正係数を表しているとすれば，位置の前後で正負の値が使用されるとしても，係数の方は負にはなり得ません．

「それは便利だ．早速 **LET** の使えるマクロ・アセンブラを買ってこよう．その代理店は…？」

ちょっと待ってください．そんなものはどこにも売っていないのです．

「話がちがうなァ．それじゃあ，どうすれば使えるのですか」

それはアセンブル時にマクロ定義の形式にしたがってアセンブラに教え込む必要があります．マクロ・アセンブラ自身は買ってきたものそのままでは機械語以外は解釈できません．マクロ定義がすべてを構成するカギになっているのです．

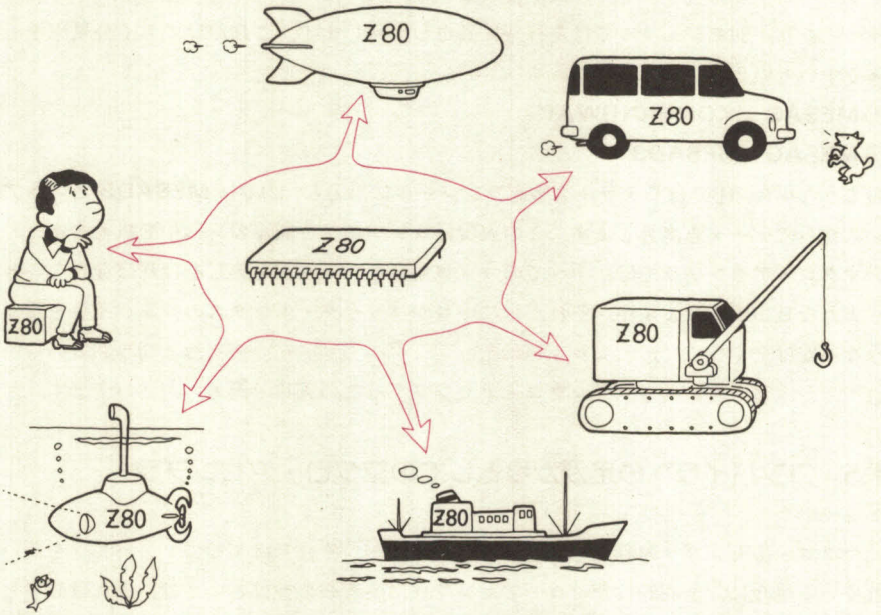
「それではコンパイラを設計しているのと同じことではないですか？一般のユーザには技術的にも時間的にも無理なのでは？」

疑問の趣旨はもっともですが，実際にはマクロ定義とコンパイラでは大きな違いがあります．それは，コンパイラではたとえ限られた専用分野のものであっても完成された体系や文法を備えていなければなりません．これに対して，マクロ・アセンブラなら必要に応じて，また，できる範囲でマクロ機能を利用することができます．マクロ定義ひとつから始めればよいわけです．コンパイラを使うには，ある時点で切り替えが必要ですが，マクロ・アセンブラはなしくずし的に高度な表現を取り入れることができ，ショックがありません．**LET** があるからといって，**GOTO** や **FOR NEXT** がなければならないという理由はありません．**JR NZ,** ～や，**LD A,** ～に混じって使ってもマクロ・アセンブラは困りません．

マクロ・アセンブラでは，何でもすべて BASIC のまねをするような使い方は得策ではありません．それよりも，適用する分野や作業内容を直接表現するマクロ命令を，読みやすく，書きやすく表現した方がはるかにメリットが出せます．これがある程度できるようになれば，高級言語は足元にも及ばないようなレベルに持って行けます．

```
ON   MAINSW, STARTLAMP
OFF  HEATER
```

のような表現を直接使えるコンパイラはありません．ユーザ定義の手続きとしてなら似たような表現が使えますが，ユーザが定義するのならマクロ・アセンブラでマクロ定義しているのと同じ手数がかかります．また，スピードはマクロ・アセンブラで定義した方が2～数倍速くなります．コンパイラではパラメータの省略や，パラメータの属性によって自

〈図 1.2〉 マクロ定義しだいでどのようにも変身するマクロ・アセンブラ

動的に処理内容を変更する手段がないものが多く，マクロ・アセンブラよりもはるかに制約が多くなります．

このような違いは，コンパイラのユーザ定義処理が実行時解釈であるのに対して，マクロ命令は翻訳時解釈になっていることからきています．これらの例として，

① **WAIT　　5, SEC**
② **WAIT　　100**
③ **WAIT**

の時間待ちマクロ命令で説明します．①は 5 秒間待つ意味で，②では単位が省略されています．単位を省略すればミリ秒を表すと約束しておきます．そして③では時間の値も省略されています．この場合は 20 ミリ秒と約束しておけばよいわけです．もちろん 20 ミリ秒に限りません．そのシステムで最もよく使われる待ち時間を③のように表し，最もよく使

われる時間の単位を②の形式で表現すればよいという意味です．

コンパイラではパラメータの数が合わないと，同一モジュール内ではコンパイル時のエラーになり，別モジュールでは実行時に暴走したりします．これだけでも十分見やすく書きやすいといえますが，さらに，

① **MESAG　'KONNICHIWA!'**

② **MESAG　MESADS**

のような場合，①ではリテラル文字をコンソールに出力し，②では**MESADS**というアドレスからのデータを出力します．この表現はコンパイラと同等のレベルで扱えます．

マクロ・アセンブラでは，すべてのマクロ命令をユーザが定義しなければならないという大きな負担がありますが，それだけに中身がブラック・ボックスのコンパイラを使うような不安は全くなく，水も漏れない緻密なプログラミングが高級言語と同等あるいはそれ以上のレベルで扱い得ます．マクロ・アセンブラはこの意味で両刃の剣といえます．

1.5　コンパイラへの足がかりとしてのマクロ・アセンブラ

マクロ・アセンブラの利点をずい分書きましたが，それではすでにコンパイラを使い慣れている場合に，全面的にマクロ・アセンブラに戻る必要性はといえば，これは必ずしもありません．この場合はコンパイラでは記述性が悪い部分やスピードが遅い部分についてのみ，アセンブラを使用すればよいといえます．多くのコンパイラは，アセンブラからの出力(オブジェクト・コード)とリンクする機能を持っていて，パラメータの渡し受けさえ正しく行えば他には何も問題はありません．

むしろ，問題はハードウェア的なレベルからプログラムを手がけたのちに，高級言語を使用していない場合の方が大きいといえます．というのも，ソフトウェアの生産性はコンパイラの方がアセンブラよりもはるかに優れているからです．部分的には確かにマクロ定義を駆使してかなりの記述性向上が図れますが，総合的に見ればやはり本格的なコンパイラにはかないません．仮に，現在は満足のいくコンパイラが存在していなくても，将来それに近いものが出現するかも知れません．そこで困るのがコンパイラの評価選択です．機械語アセンブラのレベルからは，コンパイラ的な言語感覚がつかめないので，適用する分野にどの言語が向いているのか見当がつきません．

マクロ・アセンブラで，各分野に適した表現形式を取り入れて行き，マクロ表記のレベルでプログラミングする感覚が身につけば，コンパイラ食わず嫌いはもとより，次々と発

表されるコンパイラの中に，適用可能なものが出現した時に適切な評価ができ，導入に当たっても感覚的な抵抗は全くないことに気付くはずです．

マクロ定義に頭を悩ますことは，コンパイラの設計と本質的には同じなのです．マクロ定義の手法に慣れることによって，コンパイラの気持ちがわかるようになります．筆者の考えでは，リアルタイム制御の分野でも，いずれはコンパイラを採用せざるを得なくなると見ています．コンパイラの種類も増えて行きます．ただし，必ずしも100％満足なものが出現するとはいえません．これは90％でも85％でもよいのです．残りの部分をマクロ・アセンブラで高級言語風に記述すれば十分です．

このようにして，マクロ・アセンブラは高級言語導入にあたって，そして導入後の不足分を補うについても威力を発揮し続けてくれます．

第2章
マクロ・アセンブラの基礎

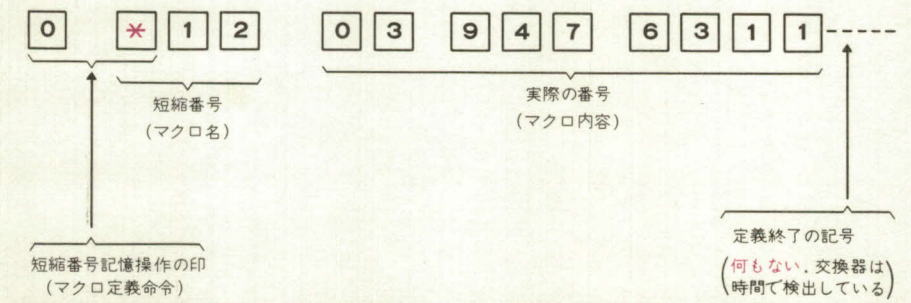
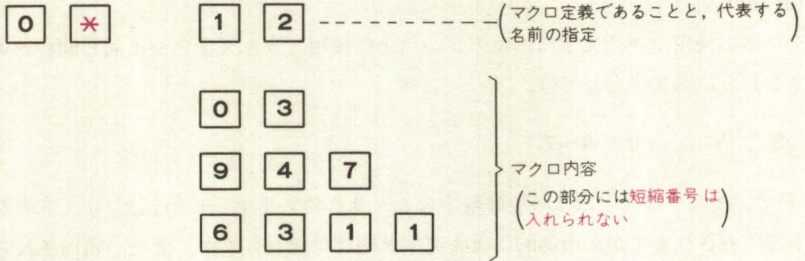
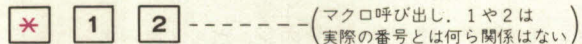
前章でマクロ機能は短縮ダイヤルのようなものだと書きました．そこで，マクロ定義を短縮ダイヤルの使用法に対応させて見てみましょう．

まず，交換器に短縮ダイヤルを記憶させなければなりません．もう少し正確にいうなら，どの電話番号を短縮番号の何番として記憶させるかを指定しなければなりません．

0＊12039476311

これがそのときの操作例です．これをマクロ定義風に分析すると図2.1のようになります．さらにアセンブラの形式に似せて書き直せば図2.2のように書けます．

マクロ・アセンブラの基本機能は短縮ダイヤルと全く同じと考えてよいのですが，この機能だけでは自由な表現形式に発展させることはできません．ちなみに，短縮ダイヤルでは実際の番号の前の部分(たとえば，03-947-63まで)だけを記憶させることはできますが，呼び出し時に残りの部分(11)を追加して正しい番号に合成するようなことはできません．また，図2.2のマクロ内容の部分に別の短縮ダイヤルを呼び出すような指定もできません．

〈図2.1〉 短縮ダイヤルのマクロ風解釈

〈図 2.2〉 図 2.1 をアセンブラ風に書き直すと

マクロ・アセンブラでは，上記の 2 点以外にも多くの機能が取り入れられており，多種多様な表現形式が解釈できるように配慮されています．本章ではマクロ定義－呼び出しの基本から，各種機能を積み上げて行く形でその応用法を実例を掲げながら解説します．

2.1　マクロ・アセンブラへの第一歩

マクロ・アセンブラの表記法は BASIC や PASCAL のように統一された規格がありません．そこで，実際に利用するに当たっては，使用するアセンブラの仕様にしたがって記述する必要があります．本書においては，MACRO 80 の Z80 フォーマットを使用し，マクロ表記に関する仕様もこれに基づいています．他のマクロ・アセンブラを使用する場合にも，ほとんど同じように使えますが表記法や制限範囲などに違いがあります．この点，各アセンブラの仕様に注意してください(MAC，RMAC とはよく似ています)．

さて，MACRO 80(以後 M80)では，他の多くのマクロ・アセンブラがそうであるように，マクロ呼び出しは命令コードでのみ可能です．前章で説明したマクロ呼び出しの **LET** や　(17および 9 ページ)も命令コードとして扱われます．マクロ定義のときに，そのマクロ内容に対して付けた名前が，そのままマクロ命令のニモニックになります．これはちょ

うど，**.COM** ファイルにつけた名前がそのままコンソールからコマンドとして使えるという関係に似ています．

マクロ名に使用できる文字は，他のシンボルに使用できる文字と全く同じ制約があります．すなわち，頭文字としては，

$.? _ @ A~Z

の31種で，後にくる文字はこの他に数字の0～9も使えます．小文字は，リストの上には小文字で出力されますが，内部的には大文字と同等に扱われます．また，識別される文字数は16文字までで，17文字以上を使用した場合は内部的には無視され，16文字までが一致すれば同一シンボルと見なされます．リスト上には17文字以後も出力されます(図2.3)．

マクロ定義とマクロ呼び出しは他のシンボルと違って前方参照ができません．すなわち，後から定義されるマクロ命令を定義に先立って使用するとエラーになります．マクロ・アセンブラは，マクロ命令を定義にしたがってその内容に置き換える(マクロを展開するという)作業を行いますが，このとき展開された形でのファイル，すなわちマクロ命令を含まないファイルを一旦作成するわけではありません．もしこれを行えば，前方参照も可能になりますが，アセンブル時間はマクロの使用量によって何10倍もかかることになるでしょう．

通常は展開作業と展開結果のアセンブル作業とが同時に行われていき，アセンブラでのいわゆるパス1(シンボル定義のパス)でもすべてのマクロが展開できなければなりません．マクロを展開しないと，その展開結果が何バイトになるか決定されませんから，それ以後の各命令のロケーションが決まらず，パス1ですべてのシンボルが定義できなくなってしまいます．

これをさけるために，マクロ命令は前方参照が許されていないのが普通です．実用上は，これが特別大きな支障になることはありませんが一応知っていなければならない大切なポイントです．アセンブラによっては，マクロ定義はすべての他の命令に先立っていなければならないものもあります．これは使いにくくなります．

前方参照ができない代わりに，同一のマクロ名を何度でも定義することができます．これを再定義可能といいますが，再定義が許されないアセンブラもあり，このあたりは個々の仕様に注意しなければなりません．

〈図 2.3〉 識別文字数のテスト

```
                                        ; ｼｷﾍﾞﾂ ﾓｼﾞｽｳ ﾉ ﾃｽﾄ
                                        ;
0001                                    ABCDEFGHIJKLMNOP        EQU     1
FFFF                                    ABCDEFGHIJKLMNOQ        EQU     -1
                                        ;
0000'   21 0001                                 LD      HL,ABCDEFGHIJKLMNOPQ
0003'   21 FFFF                                 LD      HL,ABCDEFGHIJKLMNOQPXYZ
                                        ;
                                                END

Macros:

Symbols:
0001    ABCDEFGHIJKLMNOP FFFF    ABCDEFGHIJKLMNOQ
```

2.2 マクロ定義とマクロ呼び出し

最も簡単なマクロ定義を考えてみます．その前に最も簡単なサブルーチンはどうでしょうか．

```
KANTAN:  RET
          ⋮
         CALL  KANTAN
          ⋮
```

これが最も簡単な **RET** のみのサブルーチンです．サブルーチンの場合，何もないものは作れません．必ず最低限 **RET** 命令だけは必要です．それではマクロ定義はどうなるでしょうか．

```
SIMPLE     MACRO
           ENDM
```

これで，**SIMPLE** という名前のマクロ命令が定義できます．ただし，その名の通り中身は何もありません．これは空(くう)マクロと考えられます．サブルーチンでは何もしないものはできても何もないものは作れません．

このマクロを呼び出すにはマクロ名を命令として書けばよく，

```
        :
     SIMPLE
        :
```

これでこのマクロ命令SIMPLEの位置にその内容(空(くう))が展開されます．これは機械語のNOPやLD A, Aとも違って，空ですから何も書いてないのと等価になります．すなわち，「ここには何も書いてない」と書かれているようなものです．このことは，デバッグ用に途中経過を表示させるマクロ命令を各所に入れておいても，デバッグ完了後マクロ定義の内容を消せば，各所に書き込んだマクロ命令の方は消さなくてもオブジェクト・コード上には何らの痕跡も残らないことを意味します．

もし，デバッグ用にサブルーチンを使用していれば，CALLの3バイトとCALL+RETの実行時間がサブルーチンの内容をなくしても残ることになります．もちろん，ソース・プログラム上で呼び出しているところをすべて消せば何の痕跡も残りませんが，再び入れたいとなった時，その作業量は消すほど楽ではないはずです．

```
MARK     MACRO
         DB  '-----'
         ENDM
```

このマクロはMARKと書いて呼び出された時，マイナスの文字コード5個に展開されます．これもデバッグ用に使われるマクロで，リロケーションされたプログラムの位置を見付けやすくするために入れておくものです．そして，デバッグ完了後は空マクロにして冬眠させます．また，マクロ定義の方でマイナスをプラスになど変更すれば，すべてのMARKが変更されます．このように，空マクロや1行のみのマクロでも作業性をよくする手段になります．

マクロ機能は，普通は何ステップかの繰り返し使用する命令グループを登録しておいて，必要なときに1行だけ書けば登録されたグループが再現されるといわれます．この典型的な例を掲げてみましょう．

```
HXASC    MACRO
         OR      240
         DAA
```

```
        ADD     A,160
        ADC     A,64
        ENDM
```

このマクロは,**A** の下 4 ビットを取り出して,その値を 16 進表示の ASCII コードに変換するものです. **HXASC** 命令コードが, **OR〜ADC** の 4 行に変換されます.

```
     ⋮
LD  A,10
HXASC
     ⋮
```

と書けばこの部分が実行された時, すなわち **ADC** が実行され終わった時点で **A** には **41H** (ASCII の文字 A のコード) が乗っていることになります. 16 進を ASCII コードに変換する方法は他にも種々考えられますが, 上の 4 行が 7 バイト 25 クロックで最短, 最高速です.

インタラプト処理などで, 全レジスタをスタックに **PUSH** したり **POP** したりする場合, 1 命令で全レジスタの **PUSH** ができれば便利です. ところがこれをサブルーチンにしようとすると意外にめんどうで, 実行時間もかかってしまいます. これをマクロ定義すると,

```
PUSHAL  MACRO
        PUSH    HL
        PUSH    AF
        PUSH    BC
        PUSH    DE
        ENDM
POPAL   MACRO
        POP     DE
        POP     BC
        POP     AF
        POP     HL
        ENDM
```

このようになります．これにIX, IYも含めてもよいでしょう．インタラプト処理部分では，

```
INTRPT:    PUSHAL    ;全レジスタPUSH
           :
           :      (インタラプト処理)
           :
           POPAL     ;全レジスタPOP
           EI
           RETI
```

のように使います．インタラプト処理で使うことが決まっていて，他の部分で使う可能性がないときは，レジスタの**POP**だけでなく，**EI**と**RETI**も含めてマクロにした方が便利です．この場合，**POP　HL**と**ENDM**の間に**EI**と**RETI**を書けばできあがりです．ついでにマクロの名前も**POPRET**のようにインタラプト処理が終了することを明白に表現した方がよいでしょう．

これでマクロ定義と呼び出しの方法は理解できたと思いますが，形式としていえば**MACRO**から**ENDM**までの間の行の集合(空でもよい)が，**MACRO**文に付けたシンボルを名前として定義され，その名前が命令として書かれている時，もとの行の集合を再現するといえます．

このとき，マクロ名については，使用文字などの制約の他には何の規定もありません．すなわち，.でも@でもすべてマクロ名になり得るのです．**M1**，**M2**でもよいわけで，極端には**PUSH**する機能を定義したマクロに**POPAL**と名付けてもアセンブラは正しく**PUSH**命令を作ってくれます．アセンブラは何も困りません．困るのはプログラマの方です．これまでの説明でも，すべてマクロ名はできるだけ内容を表し，しかも簡潔になるように考えて付けています．このあたりは，プログラマの創造性や言語感覚をたよりにするしかありませんが，少し後で見ると何のつもりで作ったマクロなのかさっぱり見当がつかなくて，そのつどマクロ定義を見直しているようではマクロ使用のメリットが半減してしまいます．名は体を表すように心がけましょう．

〔例題①〕　**A**の上下の4ビット(ニブル)を入れかえるマクロを定義せよ

〔例題②〕　**HL**をデクリメントして，結果がゼロならゼロ・フラグが立つようなマクロを定義せよ

〔例題③〕　**IX**をデクリメントして，結果がゼロならゼロ・フラグが立つようなマクロを定義せよ

〈図 2.4〉 例題①〜③の解答例

```
 1:  ; ﾚｲﾀﾞｲ 1
 2:  ;
 3:  SWAP      MACRO
 4:            RRCA
 5:            RRCA
 6:            RRCA
 7:            RRCA
 8:            ENDM
```

```
 1:  ; ﾚｲﾀﾞｲ 2
 2:  ;
 3:  DCHLZ?    MACRO
 4:            DEC     HL
 5:            LD      A,L
 6:            OR      H
 7:            ENDM
```

```
 1:  ; ﾚｲﾀﾞｲ 3｡ HL ｦ ｺﾜｻﾅｲ
 2:  ;
 3:  DCIXZ?    MACRO
 4:            DEC     IX
 5:            PUSH    IX
 6:            EX      (SP),HL
 7:            LD      A,L
 8:            OR      A
 9:            POP     HL
10:            ENDM
```

2.3 呼び出し時修飾――仮パラメータと実パラメータ

前節でのマクロは，各々有用な機能を持ったものですが，呼び出し時に変更できる部分は全く含まれていません．例題①でも **A** のニブルでなく，指定したメモリ・アドレスの内容のニブルの入れ替えならどうでしょうか．実行時にアドレスを指示する手段がなければ指定したいアドレスの数だけマクロ定義をしなければならないし，それならマクロにしない方が簡単です．そこで，マクロ機能には必ず呼び出し時に部分的に数値や命令を置き換える方法が用意されています．

たとえば，

MOVWRD　　ADS1，ADS2

と書いて，**ADS1** の番地から入っている 2 バイト(1 ワード)のデータが **ADS2** の番地に移せると便利です．ここで **MOVWRD** はマクロ名ですが，**ADS1** や **ADS2** は何でしょう

か．また，マクロ定義との関係はどのようになっているでしょうか．この場合のマクロ定義をわかりやすく書くと，

```
MOVWRD   MACRO    ○, △
         LD       HL, (○)
         LD       (△), HL
         ENDM
```

となります．これはもちろんアセンブラにかけることはできません．説明の都合上このように書いたものです．先の **MOVWRD** の呼び出し方でこのマクロを呼び出すと，○の所が **ADS1** に，△の所が **ADS2** に置き換えられて展開されます．

このマクロ定義にはふたつの **LD** 命令に○と△が使われています．この意味は **ADS1** と **ADS2** に置き換えるために必要なことはうなずけます．しかし，展開された結果に現れてこない **MACRO** の行に○と△が書かれている意味はあまりなさそうにも思えます．そこで，次の例を見てください．

```
IOBSET   MACRO    ○, △
         IN       A, (○)
         SET      △, A
         OUT      (○), A
         ENDM
```

これは，○のポートを読み取ってビット番号△をセットし，再び○のポートへ送り返す，つまりI/Oのビットをセットするためのマクロです．

```
IOBSET   3, 5
```

として呼び出せば，ポート3のビット5がセットされるように展開されます．なぜ，このときポート5のビット3ではなく，ポート3のビット5に決定できるのでしょうか？それは，**MACRO** の後に最初に書かれているのが○で，呼び出し時に **IOBSET** の後に最初に書かれているのが3だからです．もし，**MACRO** の行を，

```
IOBSET   MACRO   △, ○
```

と書いておけば，上の呼び出し例ではポート5のビット3がセットされるように展開され

ることになります．

このようにマクロ定義の **MACRO** から **ENDM** の間に書かれている○や△のことを仮パラメータ(またはダミー，ダミー・パラメータなど．ダミーは替え玉の意味)と呼びます．**MACRO** の行に書かれた仮パラメータは，

「この位置に書かれたものを」

というような意味を持っています．また，その次の行から **ENDM** までの間に書かれたときは，

「この位置に展開せよ」

という意味と考えられます．

ところで，○や△はアセンブラが受け付けてくれません．実際のソース・プログラム上では，仮パラメータにもシンボル名の規定にしたがった名前を付けなければなりません．ただし，このときの名前はマクロ展開のためにのみ使用されますから，**MACRO** から **ENDM** の間でさえ重複がなければエラーにはなりません．ですから，マクロ定義のたびに仮パラメータの名前を考えるのがめんどうなら，**P1**, **P2**…というような名前をどのマクロについても共通して使うこともできます．すべてのマクロが展開された時点では仮パラメータの名前は残っていません．

それなら，もっと簡単に仮パラメータに **A**, **B**, **C** のようにアルファベット１文字を使用してはどうでしょうか．

```
MOVWRD    MACRO    A, B
          LD       HL, (A)
          LD       (B), HL
          ENDM
```

これを前の例のように呼び出せば，

```
          LD     HL, (ADS1)
          LD     (ADS2), HL
```

と展開されて意図通りになります．ところが，

```
IOBSET    MACRO    A, B
          IN       A, (A)
```

```
        SET     B, A
        OUT     (A), A
        ENDM
```

と書いておいて前の例のように呼び出すと,

```
        IN      3, (3)
        SET     5, 3
        OUT     (3), 3
```

と展開されます. これでマクロ展開作業としては正常で, 展開上のエラーは発生しませんが, 機械語コードに変換する段階でエラーになります. 仮パラメータ以外にすでに **A** というシンボルが使用されているのに, 仮パラメータとして同じ名前を使用するとこのような混乱が起こります. マクロ・アセンブラの方ではレジスタ・シンボルの **A** と仮パラメータの **A** を区別する能力はありません. M80 では仮パラメータにどのような名前でも使用できる反面, 手軽に短い名前をつけると思わないところで重複して予期しないエラーが発生したり, エラーにならないのに意図通りに働かなかったりします.

ちなみに, M80 以外のマクロ・アセンブラでは, 仮パラメータには特有の頭文字を使用する規定のものや, 仮パラメータの名前が自動的に決定されているものもあります. 自動的に決定される場合は **MACRO** 行では仮パラメータを書く必要がありません. 任意の名前が使える M80 のメリットは, 大きなマクロ定義をするときに, 仮パラメータの名前にもどんな働きをするものかを表現できる点にあります. 実際, 小さなマクロでは **P1, P2** や **?A, ?B** などでも十分実用になりますから, 慣れないうちはワンパターンで通してもよいと思います.

マクロ呼び出し時に与えるパラメータを実パラメータと呼びます. **MOVWRD** の **ADS1** と **ADS2**, そして **IOBSET** の **3** と **5** がこれに当たります. このパラメータは機械語生成に直接かかわるものですから, シンボル名の場合はプログラム全体の中で定義されている必要があり, 重複は許されません. 要は直接 **LD　HL, (ADS1)** と書いたのと全く同じ制約になるわけです.

マクロの仮パラメータと実パラメータの間には, 機械語命令のパラメータのような厳しい対応制約がありません. すなわち,

① **MACRO** 行に書いたもの(仮パラメータとして使用する宣言にあたる)を, 必ずしも

〈図 2.5〉 パラメータを使ったマクロ展開例

```
                                .Z80
                        ;
                        ;   ﾏｸﾛ ﾃｰｷﾞ｡ ﾊﾟﾗﾒｰﾀ 3ｺ
                        ;
                        LD      MACRO   PP,QQ,RR
                                INC     B
                                ENDM
                        ;
                        ;   ﾏｸﾛ ﾖﾋﾞﾀﾞｼ｡ ｼﾞﾂ ﾊﾟﾗﾒｰﾀ 2ｺ
                        ;
                                LD      A,0
  0000'     04      +           INC     B
                        ;
                                END
```

ENDM までの間で参照しなくてもよい．全くのムダになるだけ．

②仮パラメータを **MACRO** 行に書き忘れてもエラーにならず，自動的に一般のシンボルと解釈される．

③仮パラメータとしてマクロ内で使用しているものでも，呼び出し時に与えなくてもよい．そのパラメータが実パラメータとしては空であると判断される．

④仮パラメータよりも多くの実パラメータを書いて呼び出してもエラーにならない．単に無視されるだけ．

となっていて，全く自由になっています．これでは簡単なケアレス・ミスやキー入力のミスでも発見できないのでは？との懸念も出てきますが，実際は機械語解釈の段階でエラーになることが多く，素通りするケースは少ないものです．ただし，エラーの内容と，その原因が結びつきにくくなるのは事実です．パラメータを与え忘れると，

LD ,( )

などというヘンなエラーが出たりして始めのうちはビックリしますが，慣れてくれば，マクロ・パラメータの段階で発生するエラーの傾向がわかり，無闇に時間がかかることはありません．図2．5はパラメータの制約がないことをいいことに，こんなマクロも作れるという例を示したものです．

〔例題④〕 アドレス **SRC** からアドレス **DST** へ **N** バイトを転送するマクロを定義せよ．呼び出しは，

TENSO　SRC, DST, N

とする．

〔例題⑤〕 アドレス **SRC** の2バイト（1ワード）をアドレス **DST** の2バイトに加算し，再び **DST** に戻すマクロを定義せよ．呼び出しは，

ADDW　SRC, DST

とする．

〔例題⑥〕 前問の加算を10進(BCD)の減算として **SUBBCD** という名前のマクロを定義せよ

〈図2.6〉 例題④～⑥の解答例

```
1:  ; ﾚｲﾀﾞｲ 4｡
2:  ;
3:  TENSO   MACRO   SRC,DST,N
4:          LD      HL,SRC
5:          LD      DE,DST
6:          LD      BC,N
7:          LDIR
8:          ENDM
```

```
1:  ; ﾚｲﾀﾞｲ 5｡
2:  ;
3:  ADDW    MACRO   A?,B?
4:          LD      HL,(A?)
5:          LD      DE,(B?)
6:          ADD     HL,DE
7:          LD      (B?),HL
8:          ENDM
```

```
1:  ; ﾚｲﾀﾞｲ 6｡
2:  ;
3:  SUBBCD  MACRO   P1,P2
4:          LD      A,(P2)
5:          LD      HL,(P1)
6:          SUB     L
7:          DAA
8:          LD      (P2),A
9:          LD      A,(P2+1)
10:         SBC     A,H
11:         DAA
12:         LD      (P2+1),A
13:         ENDM
```

2.4 マクロの中にもマクロ呼び出しが使える ——マクロ呼び出しのネスティング

前節のポートのビットをセットするマクロ **IOBSET** と全く同じ要領でセットでなくリセットするマクロが作れます．

```
IOBRES    MACRO    PORT, BIT
          IN       A, (PORT)
          RES      BIT, A
          OUT      (PORT), A
          ENDM
```

とこのように定義できますが，これは当然セットのときとほとんど同じ内容になっています．2ステップ目の命令コードだけが**SET**から**RES**に替わったにすぎません．マクロ機能が似たようなコーディングの手間を減らすためのものであれば，このように似たようなマクロ定義を簡単にやってのける手立ても用意されているはずです．

その第1の方法は，セットとリセットを分けずにひとつのマクロ命令にすることです．セットかリセットかはパラメータの方に持って行きます．この方法ではマクロの数そのものを減らしてしまうので，2つのマクロ定義をする手間を省くという言葉の意味とは若干違いますが同じ効果が得られます．その定義は，

```
IOB       MACRO    RS, PORT, BIT
          IN       A, (PORT)
          RS       BIT, A
          OUT      (PORT), A
          ENDM
```

のようにします．これを，

```
IOB    SET, 3, 5
IOB    RES, PN, BN
```

などと呼び出せば**IOB**というマクロだけでセットとリセットができます．仮パラメータ**RS**が2ステップ目の命令コードの位置に書かれているのに注意してください．マクロ機能は基本的には文字列の置き換え機能ですから，ラベルとしても命令コードとしても仮パラメータが書けます．そして，呼び出された時は実パラメータで置き換えられるわけですから，命令コードとして解釈できる文字列が与えられなければなりません．

もし，

```
IOB    ADD, PN, A
```

と呼び出せば，目的は達しませんがエラーにはなりません（**PN**は別に定義されているものとします）．しかし，

```
IOB     SUB, PN, A
```

と呼び出せばエラーになります．これは**SUB**がオペランドを1つしか使用しない命令コードだからです．このように，単に命令コードひとつだけが異なる2つのマクロは，その部分をパラメータに書き加えることによってひとつにまとめられるのです．

第2の方法はよりマクロらしい手法で，マクロ定義の中に別のマクロを呼び出すマクロ命令を書く方法です．

```
IOBSET      MACRO    PORT, BIT
            IOB    SET, PORT, BIT
            ENDM
IOBRES      MACRO    PORT, BIT
            IOB    RES, PORT, BIT
            ENDM
```

このふたつのマクロは，中にマクロ呼び出しを使わないと3ステップ必要でしたが，マクロ呼び出しを書くことによって1ステップに減りました．ただし，**IOB**というマクロが別に定義されていなければなりません．この例では第1の方法で定義したマクロをそのまま使用します．そうすると，第2の方法では3つのマクロを合計11行使って定義していることになり，**IOBSET**と**IOBRES**を直接定義した時の10行より増えてしまってメリットがありません．これは例として使ったマクロが簡単すぎたからです．もし命令の変更箇所が何個かあったら実パラメータとしてそのつど書くのは楽ではありません．

マクロ定義の中に別のマクロ命令を書く手法は，より高度に複雑化したマクロ応用では不可欠の重要な役割を果たします．

マクロ定義の中に，別のマクロ命令が書かれていると，マクロ展開中に別のマクロを展開しなければ正しく解釈できません．これをマクロ呼び出しのネスティングと呼んでいますが，サブルーチンの中から別のサブルーチンを呼び出しているのに似ています．それでは，マクロから呼び出されたマクロがさらに別のマクロを呼び出したらどうなるのでしょうか．マクロ展開中に別のマクロが呼び出されると，置き換え作業の途中でパラメータを保留したままで，次のマクロ展開作業に入り，そのマクロに展開が終わるともとのマクロ

の展開を続行します．このために，アセンブラはマクロ展開用のスタックを使用します．M80 ではマクロ呼び出しのネスティングはスタックがあふれるまで許され，スタックの大きさはオプションで拡大することができます．他のアセンブラではネスティングを許していても制限範囲はまちまちです．

いずれにしてもマクロ呼び出しのネスティングの限度は「何重まで」とはいえず，パラメータの数や使用文字数によって大きく変化するようです．

ところで，これまでの説明ではすべて，マクロの中から呼び出すのは別のマクロと限定していました．それは同じマクロつまり自分自身を呼び出すと都合が悪いからです．この点でもサブルーチンでそのサブルーチン自身を呼び出す場合と似ています．通常の単純なサブルーチンで自分自身をコールしたらスタックがあふれてしまって戻れなくなります．これを正しく機能させるには，どこかで自分自身を呼び出すことを中止しなければなりません．マクロ呼び出しもちょうど同じように，何らかの方法で呼び出しの繰り返しを中止しなければスタックがあふれてしまいます．サブルーチンのコールでスタックがあふれると暴走する可能性もありますが，アセンブル中のスタックはちゃんと管理されていて暴走の心配はありません．

さて，サブルーチンではネスティングを終わらせるのには条件ジャンプの命令で飛び出したり，条件リターンを使ったりしますが，アセンブラでのマクロ・ネスティングを終わらせるのには条件ジャンプでは効果がありません．

〔例題⑦〕 次のマクロ **NEST** の定義は正しいか．また，これを呼び出すとどうなるか確認する方法は？

```
NEST    MACRO
        DB      'NEST'
        NEST
        ENDM
```

〔解答〕 マクロ定義そのものは正しい．ただし，呼び出されるとその中から **NEST** 自身を呼び出すので

?Stack overflow，……

というエラーが表示されてアセンブルは中断される．展開される過程は存在するがパス１ですでにエラーになってしまい，リストは作成されないので，ただ単に

上記のエラーが表示されるのみでエラー箇所やエラー内容は通常確認できない.
特に,この説明のためにパス2でのみ **NEST** を呼び出すようにして作ったリストが図2.7である.これもファイルにはならないので直接プリンタへ出力する必要がある. **IF2** については次節で説明する.

2.5 アセンブラへの道しるべ――条件判断疑似命令

前節で **IOB** というマクロを定義しました. その第1パラメータは実パラメータとして **SET** または **RES** を書かなければ意図通りに働きません. しかし, これが必ずしもシステム・レベルでの動作をよく表現しているとは限りません. たとえば, そのポートの出力はすべてアクティブでLowかも知れません. とすると, **SET** でランプが消え, **RES** でつくということもあり得ます. または High で右, Low で左というのも考えられます. これを,

IOB　　ON, PLAMP, BLAMP
IOB　　LEFT, PMOT, BMOT

(**PLAMP, PMOT** はランプとモータのポート. **BLAMP, BMOT** はランプとモータのビット)

などのように書けないでしょうか.

これを解釈できるひとつの方法は **ON, OFF, RIGHT, LEFT** の4つのマクロを定義することです. **ON, LEFT** を **RES** とし, **OFF, RIGHT** を **SET** として定義しておけば目的は達せられます. 同じ内容のことを, マクロ定義を増やさないで実現できるのが条件判断疑似命令を使う方法です. 条件判断機能そのものはマクロ・アセンブラに特有のものではありませんが, マクロ定義内で使用する時, マクロ表記の柔軟性を飛躍的に拡大してくれます.

条件判断疑似命令には式の値に関するものと, 文字列の比較判定, そしてアセンブラ内部の状態に関するものの3種類が用意されています. 条件判断の機能も各アセンブラによって仕様に大きな差があります. M80では, 数値(式の評価の結果得られる数値)の判定に,

IF, IFT, COND　　式
(式の値が0でなければ…)

IFE, IFF　　式
(式の値が0ならば……)

〈図 2.7〉 例題⑦のリスト

```
                                  .LALL
                            ; Macro nesting
                            ;
                            NEST      MACRO
                                      DB        'NEST'
                                      NEST
                                      ENDM
                            ;
                                      IF2
                                      NEST                ;Yobidashi
0000'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0004'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0008'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
000C'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0010'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0014'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0018'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
001C'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0020'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0024'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0028'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
002C'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0030'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0034'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0038'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
003C'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0040'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0044'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0048'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
004C'   4E 45 53 54    +              DB        'NEST'
                       +              NEST
0050'   4E 45 53 54    +              DB        'NEST'
```

の2種類が用意されています．この式の部分には，算術四則の他，シフト，比較，論理演算などの演算子が使用でき，組み合わせによって複雑な数値判断が可能になります．

また文字列の判定用には，

IFIDN **<文字列1>，<文字列2>**

(文字列1と文字列2が同じであれば…)

IFDIF **<文字列1>，<文字列2>**

(文字列1と文字列2が異なっていれば…)

IFB **<文字列>**

(文字列が空なら…)

IFNB **<文字列>**

(文字列が空でなければ…)

の4種類が使えます．ここで各文字列はマクロの仮パラメータの名前になっているのが普通です．もし，マクロの外で，

IFB **<ABC>**

と書けば，この条件が成立するはずがありません．**IFB, IFNB** は文字列で示される仮パラメータに実パラメータが与えられたかどうかを判断するのに使用されるものです．

そしてアセンブラの内部状態に関するものは，

IFDEF **<文字列>**

(文字列で表されるシンボルが定義ずみかまたは外部に宣言されていれば…)

IF1

(パス1であれば…)

IF2

(パス2であれば…)

の4種類となっています．条件判断の書式は，

```
IF□□    □□□
  :
ENDIF
```

〈図 2.8〉 **IF** と **COND** は区別されない

```
                              ; IF and COND
                              ;
0000                          AA      EQU     0
                                      IF      AA EQ 0
0000'      01                         DB      1
                                      ENDC
                              ;
                                      COND    AA EQ 0
0001'      03                         DB      3
                                      ENDIF
                              ;
                                      IFB     <>
0002'      02                         DB      2
                                      ENDC
                                      END
```

または，

と規定されており，条件が成立した時には **IF**□□から **ENDIF** または **ELSE** までがアセンブルされ，**ELSE** があればそれ以降 **ENDIF** までは無視されます．条件不成立ではこれと逆になり，**IF**□□から **ENDIF** または **ELSE** までが無視され，**ELSE** があればそれ以降 **ENDIF** までがアセンブルされます．条件を重ねたい場合は，**IF**□□から **ELSE** の間または **ELSE** から **ENDIF** までの間に含まれる形でさらに **IF**□□~**ELSE**~**ENDIF** の組み合わせが使えます．**ELSE** がない時も同様です．では，この条件のネスティングは何重まで可能でしょうか．M80 では 255 重まで可能と規定されています．他のアセンブラでは **ELSE** が使えないものや条件のネスティングが許されていないものがあります．

ところで，**IF**，**IFT**，**COND** は全く同じ意味を持っていますが，このうち **COND** に対しては **ENDIF** でなく **ENDC** で終わるようにマニュアルには書かれています．でもこれは，内部でチェックはされません．すなわち，**ENDC** と **ENDIF** は同等に扱われていて，**IF** や **IFIDN** などすべての条件疑似命令に対して **ENDC** でも有効です(図 2．8)．

IF，**IFT**，**COND** は式の値を評価して判断しますが，これを使っても文字列の有無や定

義されているかの判断ができます．それには特別な演算子を利用します．

```
IFB     <PP>
```

は，

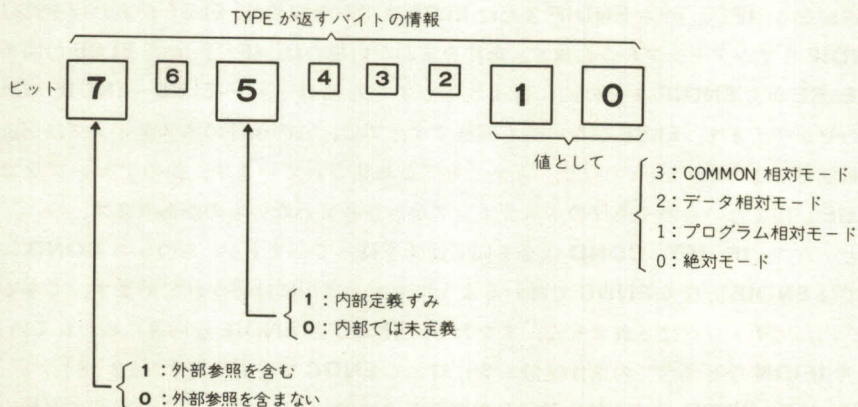
```
IF      NUL  PP
```

と同じ動作をします．**NUL** はオペランドが空白であれば**−1**（=**0FFFFH**）を返すオペレータです．このオペレータは２つ以上組み合わせては使えません．

定義／未定義を判断する演算子は **TYPE** です．**TYPE** はそのオペランドが示すシンボルがアセンブル作業上どのような状態にあるかの細かい情報を１バイトのデータとして返します．その中には，シンボルのモード（リロケータブル・アセンブラに特有のシンボルの種類．リンク時にどのエリヤの番地を割り当てるかを区別するものです）も含まれています．未定義のときは **0** が返されます．

TYPE は，単一のシンボルだけでなく，シンボルやリテラルなどの計算式の状態も判断できます．**TYPE** によって返される情報は図2.9のように各ビットに割り当てられています．

ということで，先のマクロ **IOB** のパラメータに **ON** や **LEFT** などを書けるようにするには図2.10のようなマクロ定義をすれば単一のマクロ定義で済むのです．

〈図2.9〉 **TYPE** 演算子の情報

〈図 2.10〉 マクロ **IOB** の定義

```
 1:  IOB     MACRO   RS,PORT,BIT
 2:          IN      A,(PORT)
 3:          IFIDN   <RS>,<ON>
 4:          RES     BIT,A
 5:          ENDIF
 6:          IFIDN   <RS>,<LEFT>
 7:          RES     BIT,A
 8:          ENDIF
 9:          IFIDN   <RS>,<OFF>
10:          SET     BIT,A
11:          ENDIF
12:          IFIDN   <RS>,<RIGHT>
13:          SET     BIT,A
14:          ENDIF
15:          OUT     (PORT),A
16:          ENDM
```

〈図 2.11〉 ネスティングを利用した **IOB**

```
 1:  IOB     MACRO   RS,PORT,BIT
 2:          IN      A,(PORT)
 3:          IFIDN   <RS>,<ON>
 4:          RES     BIT,A
 5:          IFIDN   <RS>,<LEFT>
 6:          RES     BIT,A
 7:          IFIDN   <RS>,<OFF>
 8:          SET     BIT,A
 9:          IFIDN   <RS>,<RIGHT>
10:          SET     BIT,A
11:          ENDIF
12:          ENDIF
13:          ENDIF
14:          ENDIF
15:          OUT     (PORT),A
16:          ENDM
```

〔**例題⑧**〕 図 2.10 のマクロ定義をネスティングを利用して図 2.11 のように変更した. 正しく働くか?

〔**解答**〕 だめ. 最初の **RS** に書かれたパラメータが **ON** であるかの判断で条件が成立した時 (**RS** が **ON** であった時) だけしか **RS** が **LEFT** であるかを判断しない. したがって, 2 番目以降の **IFIDN** は成立し得ない.

アセンブル時の演算子

アセンブラでは通常+, −, ＊, ／, **MOD, NOT, AND, OR, XOR** などの一般的な演算子が使えます. また, 比較演算子**>, <, <>, =, >=, <=**などが使えるものもあります.

M80では比較演算子は**GE, LE, NE, EQ, GE, LE**のように2文字のアルファベットで表現します. この他, **SHR, SHL**のようにシフトする演算も使えます. **LOW**と**HIGH**は2バイトの値（16ビット）を上のバイトと下のバイトに分離する機能を持っています.

HIGH　AAAは**AAA／256**と同じ結果を返し,

LOW　AAA　は　**AAA−HIGH　AAA**と同じです.

ところで, よく見ると演算子には3つの種類があります. 第1に数値をオペランドとして持って, 数値を返すもの. 第2に数値をオペランドとして, 論理値を返すもの. そして論理値をオペランドとして, 論理値を返すものです. 論理値をオペランドとして数値を返す演算子はありません.

マニュアルによれば, 論理演算子は真か偽を返すとなっています. **AND**の場合は両オペランドが真なら真を返すとなっています. ところが, 図2.Aのようなことが実際には起こります. この場合, **AA**は真でしょうか偽でしょうか？この他にも真と偽では説明のつかない例は図2.Bをはじめいくらでも出てきます.

これは, アセンブラが数値以外に論理値という属性を持っていないことから発生します. アセンブラにとっては真も偽もないのです. もう少し正しくいえば, 真という値も偽という値もないのです. それでは真偽の概念は存在しないのでしょうか. そうではありません. 真偽は**IF**□□の判断対象になった時はじめて成立する概念としてのみ存在するのです.

〈図2.A〉**AA**は真か偽か

```
0037                    AA      EQU     55
                                IF      AA AND NOT AA
                                DB      3
                                ENDIF
                                IF      AA
0000'   05                      DB      5
                                ENDIF
                                IF      NOT AA
0001'   07                      DB      7
                                ENDIF
                                END
```

〈図 2. B〉 真と偽の区別がつかない例

```
                          ; ﾛﾝﾘﾁ ｦ ｶｴｽ ｴﾝｻﾞﾝｼ ﾃｽﾄ
                          ;
0005                      AA      EQU     5
0003                      BB      EQU     3
0000'   05                        DB      BB EQ 3 AND AA
                                  END
```

〈図 2. C〉 真に 1 を加えると偽?!

```
0005                      AA      EQU     5
                                  IF      (AA EQ 5)+1
                                  DB      3
                                  ENDIF
                                  IF      (AA EQ 6)+1
0000'   07                        DB      7
                                  ENDIF
                                  IF      (AA EQ 6)-1
0001'   09                        DB      9
                                  ENDIF
                                  END
```

しかし，**IF**□□は演算子ではありません．ですから返す値がありません．ただ判断結果にしたがってアセンブルするか，無視するかを選択しているにすぎません．多くのアセンブラでは論理値の概念はありません．

それではアセンブラの説明で論理値を返すかのように書かれている演算子は本当は何をどうやっているのでしょうか．それは 2 種類あります．

①**−1 (0FFFFH)** か **0** を返すもの…>, <, =, **GE**, **LE**……などの比較演算子と **NUL**
②各ビットごとの論理演算を行うもの…**AND**, **OR** などの論理演算子

このように比較演算子については強いていえば論理的な真偽を返しているといえなくもないですが，各ビットごとの論理演算では真や偽を返しているとはいえません．とはいっても真か偽を返さなければ困るという意味ではありません．ビットごとの演算の方が利用価値がありますから，これはこの方がよいのです．要はマニュアルの説明が **AND** は両方が真なら真…とだけ説明したのでは不足だという意味です．

アセンブラの数値には論理的値(真と偽)を表すものはなく，すべて 0 ～65535(または－32768～＋32767) の数値としてのみ取り扱われているというお話でした．

コラム

2.6 アセンブル変数と文字列変換

図2.7を思い出してください．これはエラーになってしまう無限のマクロ展開なのでこのままでは使えません．けれどもここに重要なヒントがあります．それは，たった2行しか中身がないマクロ定義で無限(では困りますがそれほど長い)に文字列が作成されて行くということです．もし，これを何らかの方法で必要な長さで終わらせることができれば省力化そのものといえます．また，このままでは全く同じ文字列がただ長く並ぶだけで芸がなさすぎます．毎回少しずつ違った文字列でも作れるようならその利用価値は倍増疑いなしです．

2.6.1 アセンブル変数

ラベルに書くことで定義された値や**EQU**で定義された値，そして外部参照に指定された値は定義し直すとエラーになります．これらはアセンブル中は常に一定した値でなければならないのです．アセンブラはこの値が変わらないかを常にチェックしています．つまり，このようにして定義された値は定数として宣言されたことになります．もしこのような定数のみしか使えないとすれば図2.7のような自分自身を呼び出す手法は使えません．

そこで，**EQU**と似たような定義機能で，なおかつ何度でも定義し直すことができる代入疑似命令**DEFL**が用意されています(**MAC**では**SET**となっています．M80でも8080モードでは**SET**が使えますが，Z80モードではZ80の機械語に**SET**が使用されているので**DEFL**または**ASET**を使用します)．一度**DEFL**で定義したシンボルを**EQU**で定義し直すと多重定義エラーが発生します．これは，最初に**DEFL**で定義された時に変数として宣言されたことになるからです．

アセンブル変数という呼び名は，プログラムが実行された時に使用される変数と区別したい時に特にこのように呼ぶものです．この**DEFL**を使用すると，自分自身を呼び出しているマクロで簡単に終了させることができます．

```
NEST    MACRO
P       DEFL    P-1
        DB      'NEST'
        IF      P
        NEST
```

```
                ENDIF
                ENDM
                  :
P               DEFL        10
                NEST
```

この例では，P に10を与えて呼び出していますから，'NEST'という文字列は10回作られることになります．また，数値として9から0までのデータを発生させるのなら，DB 'NEST'を DB P に変更するだけで毎回 DB のアセンブル時の P の値がデータとして作られて，ちょうど9から0までの10個が得られます．もし1から10までが必要なら，DB P を DB 10−P とすればよく，0から45まで5ステップで欲しければ，DB 5 *(9−P) のように書けば得られることは説明の必要はないでしょう．

それでは，数値としてではなく，文字列として NEST1 から NEST10 までを発生させるにはどうすればよいでしょうか．DB 'NEST', P では P が文字コードになりませんから，DB 'NEST', 58−P とします．ASCIIコードを数値で合成するものです．これで確かに NEST1 から NEST9 までは作れますが最後がいけません．NEST : になってしまうのです．1から9はできても，桁数が変わるとこの計算法ではカバーしきれません．

2.6.2 高精度乗除算などの文字列の連結

数値処理ではいくつかのレジスタを連結してシフトする必要があります．そこで，B, C, D, E をまず1ビット・シフトするマクロを考えてみます．

```
RBCDE           MACRO
                SRA         B
                RR          C
                RR          D
                RR          E
                ENDM
```

これで右シフトはできましたが，左シフトは別に作らなければなりません．そこで，IOB の SET と RES のように，ひとつのマクロで左右両方向のシフトをパラメータによって分けようと思うと，最初の命令が SRA となっていて後の3行と違います．もし，単純にパラ

メータで命令コードを渡すとすれば，この両方を2つのパラメータで渡さなければなりません．しかし，よく考えてみると右シフトは**SRA**と**RR**，そして左シフトは**SLA**と**RL**なのです．つまり命令コードの第2文字が**R**と**L**に分かれているだけで，その他は全く共通なのです．

これを見逃す手はありません．ではどうやってその命令コードを表せばよいでしょうか．

```
SFT4        MACRO  ??
            S??A      B
            R??       C
            R??       D
            R??       E
            ENDM
```

これでは，**S??A**や**R??**は実パラメータに置き換わりません．それは**S??A**としてひとつのシンボルを表しているからです．このような場合，M80では文字列結合用の演算子&を用いるようになっています．

```
SFT4        MACRO  ??
            S&??&A     B
            R&??       C
            R&??       D
            R&??       E
            ENDM
```

これで定義は完了です．

```
      SFT4        R
```

と呼び出せば**BCDE**の右シフトが，この**R**を**L**に変えれば左シフトが呼び出されます．

このように，**&**演算子（結合子といった方がわかりやすい）はマクロの仮パラメータを参照するときのみ両側の文字列を切り離して別のシンボルと見なします．もし，

S&??&Aを**S ?? A**

と書けばこれでも別のシンボルと見なされはしますが，実パラメータで**R**が渡されても**S**

〈図 2.12〉 **&** の解釈

(a) ソース・リスト

```
 1:              .Z80
 2:              .LALL
 3:  PARA        MACRO     P1,P2,P3
 4:  P1:         P2        P3
 5:  P1A:        P2A       P3A
 6:  AP1:        AP2       AP3
 7:  A&P1:       A&P2      A&P3
 8:  &P1A:       &P2A      &P3A
 9:  P1&A:       P2&A      P3&A
10:              ENDM
11:
12:              PARA      ADS,LD,QQ
13:
14:              END
```

(b) アセンブル・リスト

```
                                         .Z80
                                         .LALL
                                 PARA    MACRO    P1,P2,P3
                                 P1:     P2       P3
                                 P1A:    P2A      P3A
                                 AP1:    AP2      AP3
                                 A&P1:   A&P2     A&P3
                                 &P1A:   &P2A     &P3A
                                 P1&A:   P2&A     P3&A
                                         ENDM

                                         PARA     ADS,LD,QQ
 U 0000'   32 0000        +      ADS:    LD       QQ
 M 0003'   00             +      P1A:    P2A      P3A
 U 0004'   00             +      AP1:    AP2      AP3
 U 0005'   00             +      A&ADS:  A&LD     A&QQ
 M 0006'   00             +      &P1A:   &P2A     &P3A
 U 0007'   00             +      ADS&A:  LD&A     QQ&A

                                         END
 7 Fatal error(s)
```

R　A となり，**S** の 1 文字が命令と解釈されてエラーになります．

それでは，実パラメータを受け取って **S&R&A** となった場合はこのような命令コードが存在しないのでエラーになりそうですが，マクロ展開が終了して機械語命令やシンボルの定義値を参照する段階では，この **&** は無視されて **S&R&A と SRA は全く同等に解釈されます．** また，連結子の **&** がクォート記号 ' か " に囲まれている中にあった場合はリスト上にもオブジェクト上にもこの **&** は現れません．アセンブラが自動的に **&** を消します．

& によって，仮パラメータの解釈のされ方がどのように変わるかをテストした結果が図 2.12 です．また，リテラル数値を **&** で結合してみたのが図 2.13 です．これによれば，数値

〈図 2.13〉 数値の連結は要注意

(a) ソース・リスト

```
 1:               .Z80
 2:   TEST        MACRO   Y
 3:               IF      Y
 4:               LD      A,9&Y
 5:               LD      A,9Y
 6:               LD      A,Y9
 7:               ENDIF
 8:               ENDM
 9:
10:               TEST    1
11:               TEST    0
12:               TEST
13:
14:   TES2        MACRO   Q
15:               R&Q&A
16:               ENDM
17:
18:               TES2    R
19:               TES2    L
20:               TES2    RC
21:               TES2    LC
22:
23:               END
```

(b) アセンブル・リスト

```
                                          .Z80
                                  TEST    MACRO   Y
                                          IF      Y
                                          LD      A,9&Y
                                          LD      A,9Y
                                          LD      A,Y9
                                          ENDIF
                                          ENDM

                                          TEST    1
  0 0000'   3E 01          +              LD      A,9&1
    0002'   3E 5B          +              LD      A,91
  U 0004'   3E 00          +              LD      A,Y9
                                          TEST    0
                                          TEST

                                  TES2    MACRO   Q
                                          R&Q&A
                                          ENDM

                                          TES2    R
    0006'   1F             +              R&R&A
                                          TES2    L
    0007'   17             +              R&L&A
                                          TES2    RC
    0008'   0F             +              R&RC&A
                                          TES2    LC
    0009'   07             +              R&LC&A

                                          END
```

(注) 本書ではソース・プログラムはすべて行番号付きで，アセンブル・リストは行番号なしで印字されています.

を連結して作るのは推奨できません.

&の記号は, 数値の途中以外はどこにあっても無視され, マクロ外で直接書いても同様に無視されます.

```
        .&Z&80
A&A&A   E&Q&U   0
        L&D     A&, &A&A&A&
        END
```

これでもエラーなく正常にアセンブルできました.

2.6.3 数値を数字に…型変換演算子

1＋1と2は同じですか? 007と7は同じでしょうか? 算数を知らない人なら一様に違うと答えるでしょう. 算数を知っていると, 同じとも違うともいえる……かも知れません. もっと正しくいえば, 数値は同じだが数字はちがうとなります. 通常は数字と数字は算術演算できません. もし無理にやれば, 9と8を加えると小文字のqになるというとんでもない結果が生じます(9もqも似たような字ですが……). 算術演算できるのは数値だけです.

ところで, 2.6.1節の**NEST**の要領で10個のエラー・メッセージを発生するマクロを考えてみます. 結果としては,

```
ERR1:    DB   'ERROR  1'
ERR2:    DB   'ERROR  2'
```

のように, 各メッセージ・データの先頭にラベルをつけたいのです. マクロ呼び出しは, **NEST**のときと同じように, **P**に**10**を代入(**DEFL**)しておいて呼び出すことにします.

```
NESTE     MACRO
P         DEFL      P-1
ERR&P:    DB        'ERROR  &P'
          IF        P
          NEST
          ENDIF
          ENDM
```

このような定義が思いつきますが，これは **P** が仮パラメータになっていないので，ふたつの**&P** は10回とも**&P** のままとなります．そうすると **ERRP** が10回ラベルに登場しますから多重定義のエラーになります．

```
ERGEN      MACRO    P
ERR&P:     DB       'ERROR   &P'
           ENDM
```

というマクロを別に作って **NESTE** の **DB** の行を，

```
           ERGEN    P
```

と変更したらどうでしょうか．

実はこれももとの **NESTE** と同じ結果を生じます．これは，**P** の数値がカウンタのように毎回カウント・ダウンされて行くのに，**P** の文字(数字ではないが数を表している文字)は変化しないからです．マクロにおいては仮パラメータと実パラメータの置き換え作業は，マクロが何重にネスティングしようと，常に文字の置き換え作業に他ならないというのがその原因です．特別な操作をしない限り，マクロ展開作業の中では **IF** のオペランド以外は数値評価が行われないのです．つまり，カウンタとして変化する数値があっても文字は変化しないわけです．

それでは特別な操作をすればよいのです．その操作が数値を数字に変える演算子を使用して可能になります．前の **NESTE** の中の **DB** の行を

ERGEN %P と書くと，予定通り10種類のエラー・メッセージに10個のラベルがついたデータが作成されます．この**%**が数値評価をしてそれを数字で表して実パラメータとしてマクロ呼び出し時に数字を渡す演算子です．

この演算子は特殊で，マクロ呼び出しの実パラメータを渡すためにしか使用できません．ついうっかりと，**ERR&%P**:…と書きたくなりますが，これは正しく働きません．

%の後にはひとつの数ではなく式が書けます．式の値が計算された上で，マクロに渡すときに数字に変わって渡されます．そのとき，数値を表現する基数が問題ですが，指定がなければ10進数で，基数変更命令 **.RADIX** で変更されていれば，その基数で表した数字に変換されます．

M80では基数は2から16までの任意の値に設定できますが，通常は10進法を使いたいので10進法のままにし，16進法は **0FFH** などのように表現します．ちなみに，基数を16

〈図 2.14〉 数値評価と文字列連結

(a) ソース・リスト

```
  1:          .Z80
  2:  TEST    MACRO   P1,P2
  3:          DB      "&P1"
  4:          DB      "&P2"
  5:          ENDM
  6:
  7:  ABC     EQU     3
  8:  BCD     EQU     2
  9:
 10:          TEST    ABC,BCD
 11:          TEST    %ABC,%BCD
 12:          TEST    %ABC+%BCD,%ABC-%BCD
 13:          TEST    %(ABC+BCD),%(ABC-BCD)
 14:
 15:  VERNUM  EQU     22
 16:
 17:  CHRNUM  MACRO   CHR,NUM
 18:          DB      "&CHR&NUM"
 19:          ENDM
 20:
 21:          CHRNUM  <Version No >,%VERNUM
 22:
 23:          END
```

(b) アセンブル・リスト

```
                                .Z80
                        TEST    MACRO   P1,P2
                                DB      "&P1"
                                DB      "&P2"
                                ENDM

 0003                   ABC     EQU     3
 0002                   BCD     EQU     2

                                TEST    ABC,BCD
 0000'  41 42 43    +           DB      "ABC"
 0003'  42 43 44    +           DB      "BCD"
                                TEST    %ABC,%BCD
 0006'  33          +           DB      "3"
 0007'  32          +           DB      "2"
O                               TEST    %ABC+%BCD,%ABC-%BCD
 0008'  30 30       +           DB      "00"
 000A'  30 30       +           DB      "00"
                                TEST    %(ABC+BCD),%(ABC-BCD)
 000C'  35          +           DB      "5"
 000D'  31          +           DB      "1"

 0016                   VERNUM  EQU     22
```

```
                              CHRNUM  MACRO   CHR,NUM
                                      DB      "&CHR&NUM"
                                      ENDM

                                      CHRNUM  <Version No >,%VERNUM
  000E'   56 65 72 73    +            DB      "Version No 22"
  0012'   69 6F 6E 20    +
  0016'   4E 6F 20 32    +
  001A'   32             +

                                      END
```

進に変更しても，**FF** では数字にならず，**0FF** と 0 が先行していなければなりません．つまり，手間はあまり減らないということです．

図 2.14 は**%**と**&**の効果を示しています．

〔**例題⑨**〕　エラー・メッセージ用のデータを作るマクロを定義せよ．ただし，ラベルは **1**，**2**，**3** のように 1 ずつ増え，メッセージの方は **1**，**2**，**4**，**8** のように 2 の n 乗になるようにすること．

```
ERR&3:    DB    'ERROR  4'
ERR&4:    DB    'ERROR  8'
                  ⋮
```

〔**解答例**〕　図 2.15 に示す．アセンブル変数を 3 つも使っているのがミソ．

2.7　繰り返しを簡単に――反復疑似命令

8 ビットのマイコンではハードウェアの持つ能力が弱いので，全く同じ命令が連続して繰り返し使われることが多いものです．また，全く同じでなくても，1 文字だけ違う命令が何個か連続することはさらに多いケースです．前節ではアセンブル変数をカウンタにして，自分自身を呼び出すマクロで繰り返し，同じまたは似たようなデータを作りましたが，これをもっと手軽にできる方法があれば重宝します．

M80 には，このような繰り返しのために反復疑似命令と呼ばれるマクロ的な命令が組み込まれています．マクロ的というのは **MACRO** を使用しても同じ機能が得られることと，書式上 **ENDM** で終わることが似ているからですが，定義と呼び出しが分かれていない点

〈図 2.15〉 例題⑨の解答例

```
                                   NESTE   MACRO
                                   P       DEFL     P-1
                                           ERGEN    %Q,%R
                                   Q       DEFL     Q+1
                                   R       DEFL     R*2
                                           IF       P
                                           NESTE
                                           ENDIF
                                           ENDM
                                   ;
                                   ERGEN   MACRO    Q,R
                                   ERR&Q:  DB       'ERROR &R'
                                           ENDM
                                   ;
 0001                              Q       DEFL     1
 0001                              R       DEFL     1
 000A                              P       DEFL     10
                                           NESTE
 0000'   45 52 52 4F      +        ERR&1:  DB       'ERROR 1'
 0004'   52 20 31         +
 0007'   45 52 52 4F      +        ERR&2:  DB       'ERROR 2'
 000B'   52 20 32         +
 000E'   45 52 52 4F      +        ERR&3:  DB       'ERROR 4'
 0012'   52 20 34         +
 0015'   45 52 52 4F      +        ERR&4:  DB       'ERROR 8'
 0019'   52 20 38         +
 001C'   45 52 52 4F      +        ERR&5:  DB       'ERROR 16'
 0020'   52 20 31 36      +
 0024'   45 52 52 4F      +        ERR&6:  DB       'ERROR 32'
 0028'   52 20 33 32      +
 002C'   45 52 52 4F      +        ERR&7:  DB       'ERROR 64'
 0030'   52 20 36 34      +
 0034'   45 52 52 4F      +        ERR&8:  DB       'ERROR 128'
 0038'   52 20 31 32      +
 003C'   38               +
 003D'   45 52 52 4F      +        ERR&9:  DB       'ERROR 256'
 0041'   52 20 32 35      +
 0045'   36               +
 0046'   45 52 52 4F      +        ERR&10: DB       'ERROR 512'
 004A'   52 20 35 31      +
 004E'   32               +
                                           END
```

はマクロとは趣が違います．すなわち，定義した時すぐに展開されます．

反復疑似命令も多重のネスティングが許されていますが，その深さは **MACRO** と合わせて，スタックがあふれるまで可能です．

2.7.1 回数指定の繰り返し…REPT

反復疑似命令 **REPT** は，反復されるべき回数を式で指定してその値の数だけ同一文字列のコピーを作り出します．

```
REPT    2+3
INC     HL
ENDM
```

これで **INC HL** が5個作られます．また，

```
REPT    4
SRA     B
RR      C
ENDM
```

とすれば **BC** を16ビット・レジスタとして，4ビット右にシフトする8ステップの機械語命令が作られます．**REPT** に与える回数の式は変数を含んでもかまいませんが，展開される時点ですでに定義されていなければなりません．

REPT を使用すれば，自分自身を呼び出すマクロを定義する必要はなく，**DEFL** で回数を与える必要もありません．ただし，**MACRO** と違って展開したい時に中身を書かなければなりません．それで，最低3行は書かなければなりません．**INC HL** を3回作りたい時は3行ですから **REPT** の使用価値はありません．もし中身が3行で，これを3回繰り返すのなら直接書けば9行で，**REPT** を使えば5行です．

REPT を使うのは，単に行数を減らすだけではありません．回数を変数や変数を含む式で指定しておいて，プログラムによってその値を変更する作業が簡単にできるようにするという効果もあります．式の値はマニュアルによれば1～65535となっていますが **0** でも有効です．つまり，**0** のときは0回展開されます．**−1** は65535として評価されます．

この **REPT** を含めて，反復疑似命令は **MACRO** の内部で使用されます．**REPT** を直接使用すれば最低3行必要ですが，これを他のマクロの中で使用すれば，そのマクロを呼び出す行は1行です．ただし，展開される内容は最大1行に書ききれる文字数で制限を受けます．この使い方の最も簡単な例を図2.16に示します．このマクロ **RPT** は繰り返し回数と1つの命令を与えて呼び出すようになっています．

〈図 2.16〉 **REPT**を使ったマクロ**RPT**

(a) ソース・リスト

```
  1:         .Z80
  2:  RPT    MACRO    KAISU,MEIREI
  3:         REPT     KAISU
  4:         MEIREI
  5:         ENDM
  6:         ENDM
  7:
  8:         RPT      5,<INC   HL>
  9:
 10:         END
```

(b) アセンブル・リスト

```
                                 .Z80
                         RPT     MACRO   KAISU,MEIREI
                                 REPT    KAISU
                                 MEIREI
                                 ENDM
                                 ENDM

                                 RPT     5,<INC  HL>
0000'   23           +           INC     HL
0001'   23           +           INC     HL
0002'   23           +           INC     HL
0003'   23           +           INC     HL
0004'   23           +           INC     HL

                                 END
```

〔例題⑩〕 図 2.16 のような方法で，3 ステップまでの命令を繰り返し作成できるようなマクロを定義せよ．

〔解答〕 図 2.17 に示す．この定義で，呼び出し時にひとつの命令しか与えない場合でも問題がないことがわかる．

2.7.2 与えたパラメータの個数によって繰り返し回数が決まる…IRP

エラー・メッセージで，**ERROR** ××の××を数値から作り出す方法を 2.6.3 節で説明しました．これを数値でなく実パラメータで与えた任意の文字列に変更したいとします．しかも，1 行のマクロ呼び出しで与えたパラメータの個数だけのラベルが同時に作り出されるように．

これも自分自身を呼び出すマクロで定義できます．この時，**DEFL** でカウントするので

〈図2.17〉例題⑩の解答例

```
                                   .Z80
                            RPT    MACRO    KAISU,M1,M2,M3
                                   REPT     KAISU
                                   M1
                                   M2
                                   M3
                                   ENDM
                                   ENDM
                            ;
                                   RPT      2,<SRA B>,<RR C>,<RR D>
0000'    CB 28     +               SRA B
0002'    CB 19     +               RR C
0004'    CB 1A     +               RR D
0006'    CB 28     +               SRA B
0008'    CB 19     +               RR C
000A'    CB 1A     +               RR D
                            ;
                                   RPT      1,LDIR
000C'    ED B0     +               LDIR
                            ;
                                   END
```

はなく，実パラメータの有無でネスティングを終了させるようにします．

```
MERR      MACRO  P1, P2, P3, ……
          IFNB    <P1>
ERR&P1    DB  'ERROR  &P1'
          MERR  P2, P3, …
          ENDIF
          ENDM
```

このマクロ **MERR** を，

```
MERR    A, B, C
```

として呼び出せば **ERRA**：　**DB　'ERROR　A'** のようにラベルとメッセージの中の文字が実パラメータによって変更され，しかも実パラメータの数だけ作られます．マクロ定義の方は **P1, P2, P3,** ……となっていますが，この仮パラメータは指定したい最大パラメータの個数だけを書いておかなければなりません．もし **P1〜P3** までしか書かないで，4個以上の実パラメータを与えるとエラーにならず，知らん顔して無視されます．

反復疑似命令 **IRP** はこれと同じ動作を定義と展開を同時に行うような感じでやってくれます．前の列は，

```
          IRP   △, ⟨A, B, C⟩
ERR&△：    DB   'ERROR  &△'
          ENDM
```

と書けば全く同じ結果になります．△の部分は **MACRO** で使用する仮パラメータと同じ意味でどんな名前でもかまいません．この例では実パラメータが各1文字ですが，ここには任意の文字列が書けます．中にブランクを含めたければ⟨**BL　A　NK**⟩のように囲みます．**IRP** は実パラメータ全体を ⟨　⟩ で囲まなければならないので，渡したい実パラメータ中にブランクが必要なら，⟨　⟩の中に⟨　⟩を書くことになります．

```
IRP  X, ⟨⟨B  L⟩, A, ⟨N  K⟩⟩
```

このようにして文字列中にブランクを含む実パラメータも渡せます．**IRP** の実パラメータに空パラメータ(ブランクでなく何もない)を与えた場合，展開する回数だけが増えて仮パラメータは消えます(空に置き換わる)．

〔**例題⑪**〕 実パラメータの数だけ展開するマクロ **MERR** を用いた場合と，**IRP** を用いた場合の展開結果に差が出るとすればどのように呼び出した時か(パラメータの数 **P1**，**P2**…は十分足りるとする．展開作業の時間は問題にしない)．

〔**解答**〕 図2.18と図2.19を参照

IFNB は空パラメータがあるのとパラメータがないのと区別がつきません．したがって，図2.19のような方法で空パラメータの分まで展開だけはするという **IRP** の動作はできません．この両者は，どちらがよいというものではなく，便利な方をそのつど選べばよいのです．

2.7.3 1文字単位で仮パラメータと置き換える…IRPC

8080のアセンブラでは，レジスタがすべて1文字で表現されています．16ビットのレジスタ(レジスタ・ペア，HLやBCなど)も1文字で表現されていますから，ほとんどすべてのレジスタを操作する命令が1文字の実パラメータによってオペランドを与えられます．**IRPC** はこのような1文字オペランドの命令を変更しながら展開するのに最適な反復疑似

〈図 2.18〉 **IRP** を使った例題⑪の解答

```
                                      .Z80
                                      IRP      X,<A,B,,,,>
                                      LD       A,X
                                      ENDM
   0000'    7F               +        LD       A,A
   0001'    78               +        LD       A,B
 Q 0002'    3E 00            +        LD       A,
 Q 0004'    3E 00            +        LD       A,
 Q 0006'    3E 00            +        LD       A,
 Q 0008'    3E 00            +        LD       A,
                                      END
```

〈図 2.19〉 **MERR** を使った例題⑪の解答

```
                                MERR     MACRO    P1,P2,P3,P4,P5
                                         IFNB     <P1>
                                ERR&P1:  DB       'ERROR &P1'
                                         MERR     P2,P3,P4,P5
                                         ENDIF
                                         ENDM
                                ;
                                         MERR     AA,BB,CC,
0000'   45 52 52 4F      +      ERR&AA:  DB       'ERROR AA'
0004'   52 20 41 41      +
0008'   45 52 52 4F      +      ERR&BB:  DB       'ERROR BB'
000C'   52 20 42 42      +
0010'   45 52 52 4F      +      ERR&CC:  DB       'ERROR CC'
0014'   52 20 43 43      +
                                ;
                                         END
```

命令です.

```
IRP  X, <B, D, H>
PUSH  X
ENDM
```

これを **IRPC** で書けば,

```
IRPC  X, BDH
PUSH  X
ENDM
```

となります. この **PUSH　B** などは 8080 のコードです. Z80 コードには使えません. 8080

ではこの他，**DAD** や **LDX** などすべてレジスタを1文字で表していますから **IRPC** の応用範囲は非常に広いようです．が，ザイログ表示では **PUSH BC** と2文字になってしまって **IRPC** を直接利用できる機会は極めて少なくなります．

とはいえ，区切り記号のない文字列を1文字単位で分離する機能はこの **IRPC** のみにしかなく，高度な表現法を取り入れるには不可欠の疑似命令といえます．したがって，多くの場合他のマクロ定義の中で使用されます．**IRPC** は解釈する段階では各文字ごとに分解しますが，与えるパラメータとしてはひとつの文字列しか受け付けません．文字列が複数個あっても，最初のもの以外は無視されます．

IRPC は扱う実パラメータがすべて1文字なので，仮パラメータも気分的に1文字にしたくなります．ところが **A** や **B** は仮パラメータ以外で使用されることが多く，何でもよいというわけには行きません．このような場合は，レジスタ名にも命令ニモニックにも条件指定（**PE** や **NZ** など）にも使われていない文字を使えば安心です．Z80 コードではユーザが定義しない限り使われない文字は **G, K, Q, W** の4文字です．**IRP** や **IRPC** の仮パラメータには，これらの中から選べば1文字でも安心です．

IRPC の使用例として，1文字で表されたレジスタ・ペアを **PUSH** するマクロを作ってみましょう．マクロ内部でまず文字列を1文字ずつに分解し，その後で各文字を判断してしかるべきレジスタ・ペアを **PUSH** する命令コードを作ります．たとえば，

```
MPUSH    ABD
```

と書けば **AF** と **BC** と **DE** が **PUSH** されるというものです．当然 **H** は **HL** を，**X** は **IX** を，**Y** は **IY** を表すという関係になります．

では **A** から **AF** を，**H** から **HL** をどのようにして作り出せばよいのでしょうか？おそらく計算ではじき出すのは無理でしょう．そこで，与えられた文字が **A** なら **AF** を **PUSH** するというように一つ一つ判定して行きます．1文字に対して6回（**ABDHXY** の6文字と比較する）で，3文字なら18回比較作業が必要になります．このマクロ定義は図2.20のようになります．

ところで，この中の **IFIDN <Q>, < >** や **PUSH**，それに **ENDIF** は6回とも全く同じで，毎回変わるのは **A** と **AF** の3文字だけです．それならこれもマクロ化できるはずです．

〔例題⑫〕 図2.20で6回出てくる **IFIDN～ENDIF** を別のマクロとして **MPUSH** を定義し直せ．

〈図2.20〉 IRPC を使用したマクロ MPUSH

```
 1:             .Z80
 2:  MPUSH      MACRO    REG
 3:             IRPC     Q,REG
 4:             IFIDN    <Q>,<A>
 5:             PUSH     AF
 6:             ENDIF
 7:             IFIDN    <Q>,<B>
 8:             PUSH     BC
 9:             ENDIF
10:             IFIDN    <Q>,<D>
11:             PUSH     DE
12:             ENDIF
13:             IFIDN    <Q>,<H>
14:             PUSH     HL
15:             ENDIF
16:             IFIDN    <Q>,<X>
17:             PUSH     IX
18:             ENDIF
19:             IFIDN    <Q>,<Y>
20:             PUSH     IY
21:             ENDIF
22:             ENDM
23:             ENDM
```

〈図2.21〉 例題⑫の解答例

```
 1:             .Z80
 2:  MMPUSH     MACRO    Q,REG,RPAIR
 3:             IFIDN    <Q>,<REG>
 4:             PUSH     RPAIR
 5:             ENDIF
 6:             ENDM
 7:  ;
 8:  MPUSH      MACRO    REG
 9:             IRPC     Q,REG
10:             MMPUSH   Q,A,AF
11:             MMPUSH   Q,B,BC
12:             MMPUSH   Q,D,DE
13:             MMPUSH   Q,H,HL
14:             MMPUSH   Q,X,IX
15:             MMPUSH   Q,Y,IY
16:             ENDM
17:             ENDM
```

〔解答〕 図2.21に示す．**MPUSH** 内部から呼び出すマクロ **MMPUSH** に仮パラメータ **Q** も与えなければならないことに注意．

〔宿題〕 **MMPUSH** も6個も並んでいる．これを **IRP** を使って3行に書き直せ．

2.8 局部シンボルを自動作成する機能…LOCAL

I/O ポートのひとつのビットが1になるまで待つというケースは，入出力ルーチンでは日常茶飯事です．これがマクロ定義の中で使われたとします．ところが，

```
×××       MACRO
            ⋮
LOOP:      IN    A,(××)
           BIT   ×,A
           JR    Z,LOOP
```

```
        ⋮
ENDM
```

このマクロは2度呼び出されるとエラーになります．マクロの中には，一度だけしか呼び出されないものもあります．2.6節のエラー・メッセージを作るマクロはその例です．しかし，一般にはマクロは何度も呼び出されるものです．上のマクロが2度呼び出されるとエラーになるのは，**LOOP** というラベルが2度定義されるからです．2度以上展開されるマクロにはこのように固定的なラベルを書くことはできません．

この例ではラベルを使わずに **JR　Z, $-4** と書けばエラーになりませんが，もっと長いルーチンの場合はラベルで書かなければやり切れません．また，長くなくても条件判断によって，途中のステップ数がそのつど変化することも考えられます．こうなるとラベルを使わなければ処理不可能ということになります．

そこでマクロ・アセンブラでは，マクロ展開するたびに自動的にラベル名が変わってつけられて行くような手段を用意しています．その扱い方は個々のアセンブラによって種々あります．M80ではラベル名を局部的に使用するための宣言をします．これが **LOCAL** 疑似命令です．**LOCAL** の行に局部的に使いたいシンボルの名前を書きます．

ここで注意しなければならないのは，局部的使用を宣言したからといってもそのシンボルがマクロの外側からは参照できない(見えないという)ものではないということです．PL/I や Pascal のような構造化された言語では局部変数はスタックの上に作られて，その手続き(サブルーチン)から返ると，その記憶場所が開放されてすぐ別の用途に使用されます．マクロ・アセンブラでの局部シンボルはこのような高級言語での局部変数とは全く別のものです．

アセンブラでの **LOCAL** 宣言は，**MACRO** の展開に先立って，宣言されたシンボルにアセンブラが自動的に作成する別のシンボルを割り当てるだけです．通常は，この割り当てられたシンボルがどのような文字列になっているかを知る方法もなく，またその必要もないので **LOCAL** というスペルを使っていますが，実際には **LOCAL** で宣言されて自動作成されたシンボルも，すべてどこからでも参照できます．さらに，その値はリストの終わりのシンボル表にも載ってきます．

ということは，自動作成されるようなシンボルを，プログラマが別に定義すると多重定義になるという問題を生じます．また，シンボル表に出てきて紙が無駄なだけでなく，アセンブル作業上もメモリ・エリヤを喰われて面白くありません．しかしながら，現在のマ

クロ・アセンブラは，ほとんどすべて局部参照のシンボルを一時的に定義して，マクロ展開後はメモリを開放するという処理をやっていません．

M80では，**LOCAL**宣言されたシンボルは，**..0000**から**..0001**，**..0002**と置き換えられて行きます．これは数字ではありません．.はシンボルの頭文字に使える立派な文字です．このシンボルの下4文字は16進でカウントされることになっていますから，このことを意識しないで**..00AB**というシンボルをどこかで定義すると，**LOCAL**で作成されたシンボルと一致してエラーになり得るわけです．

マクロ呼び出しがネスティングしているときは，中から呼び出された別のマクロで使用されているシンボルが，外のマクロで**LOCAL**宣言したものと同じであっても**LOCAL**はそこまでおよびません．別のマクロが呼び出されるたびに**LOCAL**の機能が新しく起動されます．そして，そのつど置き換え用のシンボルが変化します．

ネスティングしている内側のマクロで，どうしても外側で**LOCAL**化されたシンボルを参照したい(またはその逆で外側で**LOCAL**化されたシンボルを内側のマクロで定義したい)ときは，そのシンボルを実パラメータとして内側のマクロ呼び出しの時に与えてやれば可能です．

反復疑似命令も同じような文字列を何度か作り出しますから，その展開の1回ごとに異なるシンボルを発生させたい時があります．しかし，M80では反復疑似命令内では**LOCAL**宣言が働きません．実際にはエラーになります．

M80以外のマクロ・アセンブラでは**LOCAL**宣言ではなく，この用途専用の文字(#や?など)を使用して，自動変更されるシンボルと置き換えているものもあります．

図2.22に**LOCAL**疑似命令の使用例を示します．直接**REPT**を使うとエラーになるので，**REPT**の中から**TEST**を呼び出しています．この例で**LOCAL　QQ**がないとラベル**QQ**が3回定義されることになってエラーが起きます．

〔例題⑬〕 **LOCAL**宣言されたものは，自動的に16進の数字の前に..をつけた形でカウントされて行く．それでは..**ABCD**というシンボルは作られ得るか？

〔解答〕 理論的にはあり得る．しかし，実際にはアセンブラが使用するメモリ・エリヤが40Kバイト取れたとしても..××××の6文字が登録されるのに必要なメモリは**0ABCDH**の6倍となって，263386バイト必要であり，その前にシンボル・テーブルがフルになって中断される．事実上はあり得ない．

〈図 2.22〉 **LOCAL** 疑似命令を使用したマクロ **TEST**

```
                                     TEST    MACRO
                                             LOCAL   QQ
                                     QQ:     IN      A,(5)
                                             BIT     3,A
                                             JR      Z,QQ
                                             ENDM
                                     ;
                                             REPT    3
                                             TEST
                                             ENDM
 0000'   DB 05            +          ..0000: IN      A,(5)
 0002'   CB 5F            +                  BIT     3,A
 0004'   28 FA            +                  JR      Z,..0000
 0006'   DB 05            +          ..0001: IN      A,(5)
 0008'   CB 5F            +                  BIT     3,A
 000A'   28 FA            +                  JR      Z,..0001
 000C'   DB 05            +          ..0002: IN      A,(5)
 000E'   CB 5F            +                  BIT     3,A
 0010'   28 FA            +                  JR      Z,..0002
                                     ;
                                             END
```

〈図 2.23〉 通常の I/O ポートと RAM エリアの定義

```
  1:  CTC0     EQU     10H        13:  PIOCON   EQU    23H
  2:  CTC1     EQU     11H        14:  ;
  3:  CTC2     EQU     12H        15:           DSEG
  4:  CTC3     EQU     13H        16:           ORG    8000H
  5:  SIOAC    EQU     14H        17:  KEISU:   DS     3
  6:  SIOAD    EQU     15H        18:  BIAS:    DS     3
  7:  SIOBC    EQU     16H        19:  INDAT:   DS     3
  8:  SIOBD    EQU     17H        20:  COUNT:   DS     1
  9:  ;                           21:  POINT:   DS     2
 10:  PIOA     EQU     20H        22:  BUF2:    DS     2
 11:  PIOB     EQU     21H        23:  FLAG:    DS     1
 12:  PIOC     EQU     22H
```

2.9 非実行命令もマクロを使ってスマートに

プログラムの先頭には，通例図 2.23 のような I/O ポートと RAM のエリヤの定義が行われています．これも数が少なければ息ぬきにもなりますが，多くなってくるとエディタの作業量だけでもうんざりします．定義内容が毎回同じならこの部分を別のファイルにし

〈図 2.24〉 I/O ポートを定義するマクロ **IODEF**

```
 1: IODEF  MACRO    PORT,PNAME
 2: TEMP   DEFL     PORT
 3:        IRP      QQ,<PNAME>
 4: QQ     EQU      TEMP
 5: TEMP   DEFL     TEMP+1
 6:        ENDM
 7:        ENDM
 8: ;
 9:        IODEF 10H,<CTC0,CTC1,CTC2,CTC3,SIOAC,SIOAD,SIOBC,SIOBD>
10:        IODEF 20H,<PIOA,PIOB,PIOC,PIOCON>
11:        END
```

てインクルードすれば手間はかかりませんが，システム構成が変わった時に，その入出力が扱っている動作を適確に表現していないと，保守性と読解性が悪くなって別のところで無駄を作ってしまう恐れもあります．

通常はマクロ命令というと機械語のステップをまとめてめんどうみるものと思われがちですが，**EQU** 用にも十分な利用価値があります．

2.9.1 I/O ポート定義用マクロ

CPU の周辺用 LSI は内部にいくつかの直接アドレッシング可能なレジスタを持っていて，通常は I/O ポートの連続した番地に割り付けられます．そこで，連続したポート番号に割り当てられる名前を1行で定義できるようなマクロを考えます．図 2.23 では周辺 LSI 3個を想定していますから最悪でも3行で書けますが，この例では CTC と SIO の間にスキ間がないのでこれら8つのポートを1行で割り付けできます．

図 2.24 がその定義と呼び出しの例です．図 2.23 と全く同じ定義（ポートの）が得られます．マクロ **IODEF** は **PORT** で受け取ったパラメータを **TEMP** に代入し，**IRP** が **PNAME** で受け取った複数のポート名をひとつずつ取り出して **QQ** とし，これに **TEMP** を割り付けます．最初の展開では **QQ** は **CTC0** で，**TEMP** の値は **10H** です．**TEMP** は **IRP** の展開ごとに＋1され，連続したポート番号を作り出します．

〔例題⑭〕 図 2.24 では呼び出しの時にポートにつけた名前を〈 〉で囲まなければならない．この〈 〉を書かなくても済むようなマクロ定義を考えよ．

〔解答〕 図 2.25 参照．図 2.24 では入っていなかった **IFNB** が図 2.25 には入っている．この意味は図 2.18 にある通り **IRP** が **,,** だけでも展開され，

EQU **TEMP**

〈図 2.25〉 例題⑭の解答例

```
 1:           .Z80
 2:  IODEF    MACRO    PN,P1,P2,P3,P4,P5,P6,P7,P8
 3:  TEMP     DEFL     PN
 4:           IRP      QQ,<P1,P2,P3,P4,P5,P6,P7,P8>
 5:           IFNB     <QQ>
 6:  QQ       EQU      TEMP
 7:           ENDIF
 8:  TEMP     DEFL     TEMP+1
 9:           ENDM
10:           ENDM
11:
12:           IODEF    10H,XA,XB,XC,XD,XE,XF
13:           IODEF    20H,YA,YB
14:           END
```

というエラー行を作成してしまうのを防いでいる.

2.9.2 RAM のエリヤ確保と変数名割り付け

I/O ポートは，すべての名前が1番地のみを占有しました．それで **TEMP+1** で行えたわけですが，RAM 内のデータは1バイト長とは限らず，文字列まで含めれば数十バイトという場合もあり得るでしょう．したがって，I/O ポートの割り付けのようにすんなりとは行きません．また，I/O ポートは，ハード的に決定されたものに一致した定義が必要でしたが，RAM の番地は重複したり無駄が出たりさえしなければ，それ以上に細かい制約は受けませんから，I/O ポートより融通がききます．

マクロの作り方としては，同じバイト数を占有するデータ同士をひとまとめにして，1行に書けるようにします．占有バイト数の違いは，パラメータとして与える方法とマクロ名で分ける方法とが考えられます．マクロ定義の手間からいえば，バイト数をパラメータで与える方がマクロ定義1回だけで楽にできます．

図 2.26 のマクロ **RAMDEF** がその例です．呼び出し例としては，図 2.23 のものより変数名の数を増していますが3行で書けます．このリストは展開状態を示すために展開された **DS　1** などを出力していますが，実用上は最後の **Symbols**：の欄に定義された値が出てきますから展開出力は不要です．

〔例題⑮〕 データ長1から4までのエリヤ確保用マクロをそれぞれ **BYTE**，**WORD**，**TRIP**，**QUAD** という名前で定義せよ．

〔解答〕 **RAMDEF** の仮パラメータ **BYT** をなくして，**DS　BYT** を **DS　1** から **DS　4**

〈図 2.26〉 RAM エリアを確保するマクロ **RAMDEF**

```
                    RAMDEF   MACRO    BYT,NAME
                             IRP      QQ,<NAME>
                    QQ:      DS       BYT
                             ENDM
                             ENDM
                    ;
0000'                        ASEG
                             ORG      8000H
                    RAMDEF 3,<KEISU,BIAS,INDAT,BUF3,TRIPL,ETC>
8000        +       KEISU:   DS       3
8003        +       BIAS:    DS       3
8006        +       INDAT:   DS       3
8009        +       BUF3:    DS       3
800C        +       TRIPL:   DS       3
800F        +       ETC:     DS       3
                    RAMDEF 2,<POINT,BUF2,ADRS,ETCETC>
8012        +       POINT:   DS       2
8014        +       BUF2:    DS       2
8016        +       ADRS:    DS       2
8018        +       ETCETC:  DS       2
                    RAMDEF 1,<COUNT,FLAG,BYTE1,EETTCC>
801A        +       COUNT:   DS       1
801B        +       FLAG:    DS       1
801C        +       BYTE1:   DS       1
801D        +       EETTCC:  DS       1
                             END
```

の 4 種類とし，各々に上記の名前をつければよい(図 2.27)．

この例題では特に最も細かく展開過程を出力する **.LALL** と，展開された部分を出力しない **.SALL** のリスト制御疑似命令を使用して 2 種類のリストを作りました．マクロのテストには **.LALL** のリストが必要ですが，マクロが完成したら **.SALL** にしてリストの面積を減らす，その感じがつかめると思います．

2.9.3 I/O ポートの各ビットごとに独立した名前を付ける方法

これはマクロ・アセンブラの機能というものではありません．アセンブラに組み込まれている機能を活用することで疑似的に各ビットに名前を付ける手法です．これをマクロで定義してマクロ呼び出しで参照すると，あたかもアセンブラがビット単位のシンボル定義を受け入れているかのように使用できます．

まず，図 2.1 の **IOBSET** を思い出してください．これはその名の通り I/O の 1 ビットをセットするためのマクロでした．呼び出しにあたっては 2 つのパラメータを与えています．

〈図 2.27〉 例題⑮の解答例

(a) **.LALL**によるアセンブル・リスト

```
                           .Z80
                           .LALL
                  BYTE     MACRO    XX
                           IRP      JJ,<XX>
                  JJ:      DS       1
                           ENDM
                           ENDM
                  WORD     MACRO    XX
                           IRP      JJ,<XX>
                  JJ:      DS       2
                           ENDM
                           ENDM
                  TRIP     MACRO    XX
                           IRP      JJ,<XX>
                  JJ:      DS       3
                           ENDM
                           ENDM
                  QUAD     MACRO    XX
                           IRP      JJ,<XX>
                  JJ:      DS       4
                           ENDM
                           ENDM
                           BYTE     <B1,B2,B3,B4>
              +            IRP      JJ,<B1,B2,B3,B4>
              +   JJ:      DS       1
              +            ENDM
0000'         +   B1:      DS       1
0001'         +   B2:      DS       1
0002'         +   B3:      DS       1
0003'         +   B4:      DS       1
                           WORD     <ADRS1,ADRS2,ADRS3>
              +            IRP      JJ,<ADRS1,ADRS2,ADRS3>
              +   JJ:      DS       2
              +            ENDM
0004'         +   ADRS1:   DS       2
0006'         +   ADRS2:   DS       2
0008'         +   ADRS3:   DS       2
                           BYTE     <Y1,Y2,Y3>
              +            IRP      JJ,<Y1,Y2,Y3>
              +   JJ:      DS       1
              +            ENDM
000A'         +   Y1:      DS       1
000B'         +   Y2:      DS       1
000C'         +   Y3:      DS       1
                           TRIP     <DATA,DATB,DATC,DATY,DATZ>
              +            IRP      JJ,<DATA,DATB,DATC,DATY,DATZ>
              +   JJ:      DS       3
              +            ENDM
000D'         +   DATA:    DS       3
0010'         +   DATB:    DS       3
0013'         +   DATC:    DS       3
0016'         +   DATY:    DS       3
0019'         +   DATZ:    DS       3
                           QUAD     <PAR1,PAR2>
              +            IRP      JJ,<PAR1,PAR2>
```

```
                         +     JJ:     DS      4
                         +             ENDM
001C'                    +     PAR1:   DS      4
0020'                    +     PAR2:   DS      4
                                       WORD    <POINT1,POINT3>
                         +             IRP     JJ,<POINT1,POINT3>
                         +     JJ:     DS      2
                         +             ENDM
0024'                    +     POINT1: DS      2
0026'                    +     POINT3: DS      2
0028'                          DATEND  EQU     $
                                       END
```

(b) **.SALL**によるアセンブル・リスト

```
                                       .Z80
                                       .SALL
                         +     BYTE    MACRO   XX
                         +             IRP     JJ,<XX>
                         +     JJ:     DS      1
                         +             ENDM
                                       ENDM
                         +     WORD    MACRO   XX
                         +             IRP     JJ,<XX>
                         +     JJ:     DS      2
                         +             ENDM
                                       ENDM
                         +     TRIP    MACRO   XX
                         +             IRP     JJ,<XX>
                         +     JJ:     DS      3
                         +             ENDM
                                       ENDM
                         +     QUAD    MACRO   XX
                         +             IRP     JJ,<XX>
                         +     JJ:     DS      4
                         +             ENDM
                                       ENDM

                         +             BYTE    <B1,B2,B3,B4>
                         +             WORD    <ADRS1,ADRS2,ADRS3>
                         +             BYTE    <Y1,Y2,Y3>
                         +             TRIP    <DATA,DATB,DATC,DATY,DATZ>
                         +             QUAD    <PAR1,PAR2>
                         +             WORD    <POINT1,POINT3>
0028'                          DATEND  EQU     $
                                       END
Macros:
BYTE            QUAD            TRIP            WORD

Symbols:
0004'   ADRS1           0006'   ADRS2           0008'   ADRS3
0000'   B1              0001'   B2              0002'   B3
0003'   B4              000D'   DATA            0010'   DATB
0013'   DATC            0028'   DATEND          0016'   DATY
0019'   DATZ            001C'   PAR1            0020'   PAR2
0024'   POINT1          0026'   POINT3          000A'   Y1
000B'   Y2              000C'   Y3
```

ただひとつのビットをセットするのにパラメータがふたつ．これは面白くありません．しかし，アセンブラもハードウェアも単一のパラメータでビットとポートを指定するようには作られていません．

各ポートのビットが独立した意味を持っている制御用システムでは，各々に単独のシステム上意味のある名前をつけたいものですが，ポートは **IN** 命令で，ビットは **SET** 命令で使用するので止むを得ません．それでもあきらめないでよく分析すると，ポートは256個までで，ビットは8個で合計264ではなく2048個しかありません．ということは，アセンブラの内部で扱う数値としては単一のシンボルに定義できるはずです．

たとえば，**REDLMP** がポート5のビット3に接続されているとしたら，

```
REDLMP    EQU    503H
```

とすればよいでしょう．または **53H** でも **305H** でもかまいません．本書では，このように単一のシンボルがポートとビット番号を合成した値で与えられているとき，**PBN** と書いて表すことにします．つまり，今は **PBN** の定義方法を説明していることになります．また，上の **REDLMP** は **PBN** 形式で **EQU** されているともいいます．

ところで，**PBN** の **EQU** はできましたが，たとえば **IOBSET** のようなマクロの中で，**PBN** をどのように料理すれば正しい命令コードが生成されてくるでしょうか．それにはいろんな方法がありますが，手っ取り早いのは数値の上下バイトを分離する演算子 **HIGH** と **LOW** を使う手です．

```
IOSET     MACRO     PBN
          IN        A, (HIGH  PBN)
          SET       LOW  PBN, A
          OUT       (HIGH  PBN), A
          ENDM
```

このように定義しておいて，

```
          IOSET    REDLMP
```

として呼び出せば，ポート5のビット3が1になるような命令コードが作り出されるわけです．マクロ **IOSET** の仮パラメータ **PBN** は，××でも?でもかまいませんが，特別な定義をしたシンボルを扱う場合，そのシンボルの性格(属性と考えてもよい)を表したものに

〈図2.28〉 **PBN** の定義を行うマクロ **PBNDEF**

```
PBNDEF  MACRO   PORT,BNAME
TEMP    DEFL    0
        IRP     Q,<BNAME>
        IFNB    <Q>
Q       EQU     PORT*256+TEMP
        ENDIF
TEMP    DEFL    TEMP+1
        ENDM
        ENDM
;
PBNDEF 5,<BRAKE,CLUTCH,,REDLMP,GRNLMP,PUMP,AIR>
```

しておいた方が見やすく，ミスも減ります．仮パラメータに任意の名前が使用できることは，手の込んだことをやろうとすると非常に有難いものであることがわかります．

このように，**PBN** 形式で **EQU** されたシンボルと，**PBN** の値を解釈するように定義されたマクロを組み合わせると，ポート番号に名前をつける必要はなく，各ビットに直接独立した名前がつけられているものとして操作取り扱いができるようになります．

それでは，各ビットを **503H** や **802H** のように **EQU** する必要はなくなりませんか？表面上の手間を省くにはこの **EQU** にマクロを使うことによって簡単に表現すればよいのです．そこで，**PBN** を **EQU** するためのマクロ **PBNDEF** を考えましょう．一度に定義できるビットは，同一のポートに限定し，最大は8個(すなわち全部のビット)までとすれば，図2.28のようにして定義できます．

呼び出しには，最初のパラメータとしてポート番号を与え，その後に0から順にそのビットにつけたい名前を書きます．名前をつけないビットの場所は空パラメータにしておきます．これで **REDLMP** は **503H** に定義されています．

先のマクロ **IOSET** を **ON** と名前変更すれば，

```
ON      AIR
OFF     CLUTCH
```

のようにシステム上の動作を直接表現でき，プログラムの大きな流れが非常につかみやすくなります．ついでに，

```
ON      <AIR, PUMP>
```

と **ON** したいビット名をいくつか並べて書けるとさらに使いやすくなります.

〔例題⑯〕 上記のマクロ **ON** と, これと同様でビットを0にする **OFF** を定義せよ.

〔解答例〕図2.29に示す. 展開例を示すために図2.28の **PBNDEF** を利用している.

〈図2.29〉 例題⑯の解答例

```
                    PBNDEF  MACRO   PORT,BNAME
                    TEMP    DEFL    0
                            IRP     Q,<BNAME>
                            IFNB    <Q>
                    Q       EQU     PORT*256+TEMP
                            ENDIF
                    TEMP    DEFL    TEMP+1
                            ENDM
                            ENDM
                    ;
                    ON      MACRO   PBNS
                            IRP     PBN,<PBNS>
                            IN      A,(HIGH PBN)
                            SET     LOW PBN,A
                            OUT     (HIGH PBN),A
                            ENDM
                            ENDM
                    OFF     MACRO   PBNS
                            IRP     PBN,<PBNS>
                            IN      A,(HIGH PBN)
                            RES     LOW PBN,A
                            OUT     (HIGH PBN),A
                            ENDM
                            ENDM
                    ;
                    PBNDEF 5,<BRAKE,CLUTCH,,REDLMP,GRNLMP,PUMP,AIR>
                    PBNDEF 15,<,,WATER,,,MOTOR1>
                    ;
                            ON      <MOTOR1,REDLMP>
0000'   DB 0F   +           IN      A,(HIGH MOTOR1)
0002'   CB EF   +           SET     LOW MOTOR1,A
0004'   D3 0F   +           OUT     (HIGH MOTOR1),A
0006'   DB 05   +           IN      A,(HIGH REDLMP)
0008'   CB DF   +           SET     LOW REDLMP,A
000A'   D3 05   +           OUT     (HIGH REDLMP),A
                            OFF     <BRAKE,PUMP,WATER>
000C'   DB 05   +           IN      A,(HIGH BRAKE)
000E'   CB 87   +           RES     LOW BRAKE,A
0010'   D3 05   +           OUT     (HIGH BRAKE),A
0012'   DB 05   +           IN      A,(HIGH PUMP)
0014'   CB AF   +           RES     LOW PUMP,A
```

```
0016'   D3 05          +        OUT    (HIGH PUMP),A
0018'   DB 0F          +        IN     A,(HIGH WATER)
001A'   CB 97          +        RES    LOW WATER,A
001C'   D3 0F          +        OUT    (HIGH WATER),A
                            ;
                                END
```

PBN 形式で定義して利用できるのは上記のような出力命令だけではなく，入力のビットを見て判断するのにも使えます．

```
WHEN    MACRO    PBN, POL, ADS
        IN       A, (HIGH  PBN)
        BIT      LOW  PBN, A
        JP       POL, ADS
        ENDM
```

と定義しておいて，

```
WHEN    LOCK, Z, SHORI 1
```

と呼び出せば **LOCK** が0なら処理1へ飛べという意味を表します．**LOCK** は **PBN** 形式の入力ビット，**Z** は **JP** 命令の条件指定(この場合は **Z** か **NZ**)を書きます．

このマクロの定義内容を知っていれば，

```
WHEN    AIR, NZ, $-4
```

と書けばビット **AIR** が0になるまで，この命令の位置でループするという意味になることがわかるでしょう．

〔**宿題**〕　このときのパラメータ **Z, NZ** が表現として面白くないので **ON** と **OFF** で示したい．**WHEN** を改造して定義し直せ(ヒントは **IOB**)．

2.10　反復疑似命令の展開を中断する疑似命令…EXITM

図2.25で使用しているマクロ **IODEF** の中に，**IFNB　<QQ>** という行があります．この行は実パラメータの数が8個に満たないときに，**EQU　TEMP** をアセンブルさせな

いためのものでした．これでエラーの発生はなくなります．しかし，**IRP**が8回展開されることには変わりありません．ということはアセンブル時間が8回分費されることを意味します．この例では**EQU**と**DEFL**の2行ですから8回分といってもたいした時間ではないでしょう．でも展開される行数が何百行もあったり，多重に他のマクロを呼び出す部分を含んでいたらどうでしょうか．また，最大の展開回数も**REPT**の場合は数百回という可能性もあり得ます．

そこで，ある特定の条件で展開を中断して，アセンブル時間を節約できれば作業性がよくなるわけですが，このために使用されるのが**EXITM**疑似命令です．**EXITM**は，これ自身には条件判断の機能はありませんから，通常は**IF××**と組み合わせて使用します．図2.25の**IFNB**からの3行を，

```
            IFB      <QQ>
            EXITM
            ENDIF
QQ          EQU      TEMP
       ⋮
```

の4行に変更すれば，実パラメータに空が現れた時**EXITM**がアセンブルされ，**IRP**の展開が中断されます．

中断される動作を少し細かく見れば，何回か繰り返されて展開している途中で**EXITM**がアセンブルされた(条件成立の命令コードとして検知された)時，その回の**EXITM**の後**ENDM**までのアセンブルをせず，さらに残りの繰り返しも行わないといえます．

EXITMは**MACRO**の展開も中断できます．しかしながら，**MACRO**内での使用は多くの場合**ELSE**と同じ働きになり，反復疑似命令との組み合わせほどの効果はありません．

```
EX          MACRO    Q
            IF       Q  EQ  3
            DB       8
            ELSE
            DB       0
            ENDIF
            ENDM
```

〈図 2.30〉 **REPT** を途中で中断する

```
0000                        AA     DEFL    0
                                   REPT    10000
                                   DB      55
                                   IF      AA EQ 3
                                   EXITM
                                   ENDIF
                                   DB      AA
                            AA     DEFL    AA+1
                                   ENDM
 0000'    37          +            DB      55
 0001'    00          +            DB      AA
 0002'    37          +            DB      55
 0003'    01          +            DB      AA
 0004'    37          +            DB      55
 0005'    02          +            DB      AA
 0006'    37          +            DB      55
```

このマクロ定義は **ELSE** からの3行を，

```
EXITM
ENDIF
DB      0
```

と書き直しても全く同じ働きをします．

EXITM は必ず直後に **ENDIF** を伴います．**EXITM** は条件成立でそれ以後 **ENDM** までを無視しますから，**EXITM** の後の **ENDIF** までの間に何を書いてもアセンブルされる機会は永久に起こりません．

図 2.30 は **REPT** を指定回数以下で中断する例を示したものです．**AA** がカウントされて3になった時成立する条件下に **EXITM** があり，その時の **DB　AA** は作成されません．それで **DB　55** が4回なのに **DB　AA** が3回しか作られないのです．

〔例題⑰〕 **IRP** を使ってパラメータの数だけの **DB** 行を作るマクロで，実パラメータに **A** 1文字のものがあればそれ以後の展開をしないようにした．

```
IRP     Q, <PARAM>
DB      Q
```

```
IFIDN    <Q>, <A>
EXITM
ENDM
  :
```

条件成立で **EXITM** を有効にし，条件不成立では他に何も作らないので **ENDIF** は省略した．ところがこれが予定通り働かない．その理由は何か．

〔解答〕 条件判断をするからには条件不成立もあり得る．この場合は，不成立で展開続行だから不成立の方が多いのが普通と思われる．ところが，不成立のときは，**ENDIF** までを無視することになっている．**DB** が1個作られたあとは，すべての機械語命令は作成されず，マクロも展開されない．他に **ENDIF** がポツンと現れる可能性はないので，それ以後プログラムの終わりまですべて不成立の条件下に入ることになる．条件下の **EXITM** は必ず **ENDIF** と重ねて使用しなければならない．

〔宿題〕 無条件で **EXITM** を使用する効果はあり得るか？

☆ ☆

以上，マクロ・アセンブラおよびM80を利用する上での基本機能について説明しました．以上の他にもマクロ展開リスティング制御，偽条件部リスティング制御などマクロ利用に関係の深い機能が組み込まれています．リストの中にはこれらの機能を利用しているものがありますから参考になると思います．

第 3 章
マクロ応用の基本ノウハウ

前章においては，マクロ・アセンブラおよびM80に組み込まれている機能について説明しました．マクロ・アセンブラといえども言語の一種と考えられます．言語ではいわゆる「語感」をつかむのが自由に操るコツです．アメリカ人は子供でも英語をしゃべり，日本人は子供でもBASICを使います！見慣れ，書き慣れればマクロ・アセンブラも鼻唄まじりで自由自在……になりたいものです．

そうなるために，本章ではより応用範囲が広く，より利用価値が高く，多少手の込んだ使用法を解説します．使用する疑似命令はすべて前章で解説したものですから，読み飛ばして次章へ**JUMP**しても構いません．次章に入って手強いようでしたら本章へ**RET**してください．

3.1 読み取れない出力ポートのビット・コントロール

多くのハードウェアでは，出力ポートのデータを読み取ることができません．パラレル入出力のマイコン周辺LSIでも制御用ポートを読み取る機能はなく，現在どのようなモードで使用されているのかは当のLSI自身からは知り得ないのです．その他のLSIも，出力時は制御用ポート，入力時はステータス・ポートとなっていて，やはり制御用ポートの内容は読めません．

出力ポートが読めない理由は，ハードウェアを簡単にするためと入出力で同一番地を別の機能に利用してポート数を節約するためですが，ポートが読み取れないと2.9.3節の**IOSET**のようなマクロは使えません．折角のマクロや**PBN**が特定のハードウェアでしか使えないのは残念です．

多くのシステムでは読めない出力ポートを使用しています．そして，ビット制御用にはRAMのコピーを利用して，出力したデータ内容が読み出せるようになっています．しかし，これでは困ります．RAMのバイト数が減るから……ではありません．ビットごとに名

前をつける **PBN** の他にメモリ・アドレスの名前も必要になり，**PBNDEF** のマクロも使えなくなります．何とかメモリの名前も含めて **PBN** 形式で指定できないものでしょうか．

最も簡単な方法としては，メモリ内のポート番地の最大値までのバイトをポート出力用のバッファに使用する方法です．もし，ポート 255 が使用されていれば，256 バイトが RAM 上で占有されてしまうという不都合がありますが，ポート定義もマクロも簡単にできます．この方法ではビットに名前をつける **PBN** そのものと，**PBNDEF** のマクロは変更せずそのまま使えます．RAM のエリヤ上に，

```
IOBUF:    DS    MAXIOP+1
```

として **IOBUF** から確保しておきます．**MAXIOP** は，内容を読みたい出力ポートの最大のポート番地です．

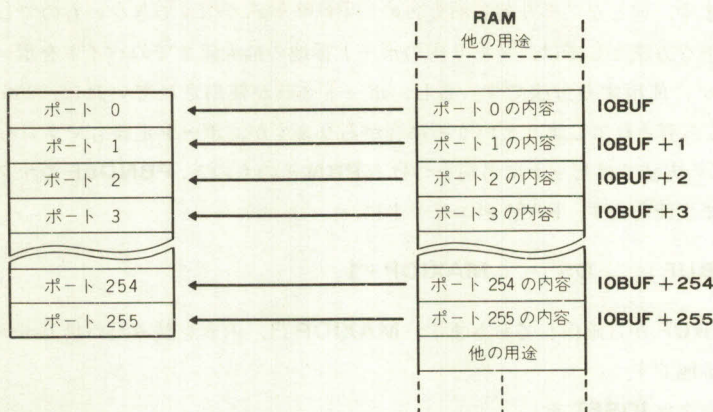
これでマクロ **IOSET** を，

```
IOSET2    MACRO  PBN
          LD   A, (HIGH  PBN+IOBUF)
          SET  LOW  PBN, A
          LD   (HIGH  PBN+IOBUF), A
          OUT  (HIGH  PBN), A
          ENDM
```

と変更します．このマクロの呼び出し方は **IOSET** と全く同じです．ただし，その動作はポートを **IN** によって読み取るのではなく，**LD A,** でメモリ内のしかるべきアドレスから持ってきて目的のビットをセットしてその番地へ帰します．さらに，これと同じデータを目的のポートへ出力します．

この方法はビット制御の出力ポートが I/O ポートのうち大部分を占めているときに適しています．また，RAM エリヤに余裕があればこの方法で十分です．では，もしビット制御の出力ポートがポート 0 とポート 255 で，RAM エリヤが 256 バイト使えない場合はどうすればよいでしょうか．それには **IOBUF+1** から **+254** までの不使用番地を他の用途に回せばよいでしょうが，その後でポート 100 あたりを追加するような時は困るハメになりそうです．

そこで，ポート番地の他にそのポートが RAM 上のどの場所に置かれているかを知るための数値 **PBF**(Port BuFfer) を導入します．このとき，もし直接 16 ビットのアドレス値を

〈図 3.1〉 **IOSET2** が必要とする RAM エリア (最大 I/O ポートが 255 のとき)

PBF で表現すれば簡単ですが，**PBN** と **PBF** を指定しないと **IOSET** のようなマクロは使えないようになります．また，ひとつのビットを単一シンボルで表すという手法がくずれてしまいます．もちろん，**PBNDEF** も，**PBN** と **PBF** を定義するように変更しなければなりません．

ところで，今，ビット制御のポートが比較的少ない場合を問題にしていますから，ついでにこの数を 31 ポートまでと限定したとします．これはポート番地の範囲ではなくあくまでもビット制御出力ポートの総数とします．この程度の量で足りる用途も多いとしてこれで考えてみます．

これまでの **PBN** は，下のバイトのうちの下 3 ビットだけしか使っていません．上のバイトはポート番地ですから，すでに空きビットはありません．つまり，下のバイトの上 5 ビットが常に 0 になっているのです．これを利用しない手はありません．

これを実現するには，**PBNDEF** と **IOSET2** を変更しなければなりません．まず，**PBNDEF** では，そのポートが RAM コピーを必要とするか，ただビット名を **PBN** 形式で決めるだけなのかを指定する方法が用意されなければなりません．この機能がないと，**PBNDEF** で定義されたポートは，すべて RAM を使用することになります．もっとも，読み取れる出力ポートを含めても 31 ポート内に納まり，しかもその程度に RAM を使用するのが差し支えなければすべて RAM を使用してもかまいません．すべてが RAM を使用するとすれば，**IOSET** も条件判断なしで定義できます．

〈図 3.2〉 **PBF** 使用のメモリとポートの対応

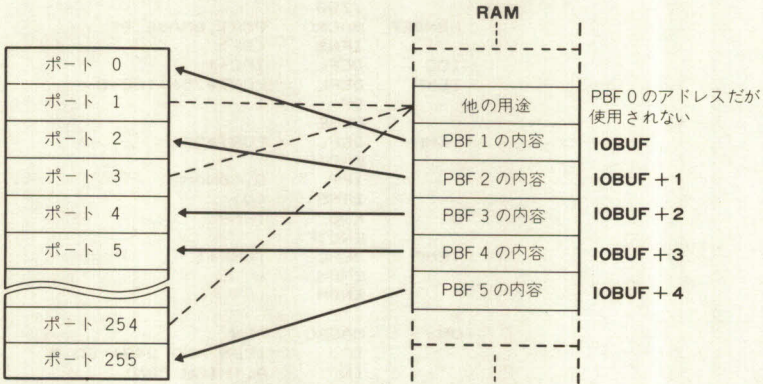



図 3.2 は，この方法でのポートと RAM の対応関係を示しています．図 3.1 に比べて必要な分だけ RAM を使用するのでエリヤの無駄はありません．

この方法に対応するマクロ定義とその呼び出し例のリストが図 3.3 です．**PBNDEF** では第 3 パラメータの有無を **IFNB** で調べ，もし空でなければ RAM を使用するポートと判断してメモリ・エリヤを **DS 1** で取ると同時に **PBF** のカウンタ **IOC** を下のバイトの上 5 ビットに加えます．もし，第 3 パラメータが空なら，これまでの **PBN** と同じ値になるようにしてあります．**IRP～ENDM** までの定義は変わりません．

一方，これを参照してビットをセットするマクロは，RAM から読み出してセットするのか，I/O ポートから読み取ってもよいのか判断しなければなりません．このために **PBF** 値が 0 （以前の **PBN** と同じ）の場合は直接 I/O ポートを読むようにします．図 3.3 ではこのマクロを **ON** という名で定義しています．**PBF** はビット 7 ～ 3 の 5 ビットに入れるために，**PBNDEF** では＊ **8** され，**ON** では/**8** で元に戻しています．**IOBUF－1** の－**1** は **PBF＝0** を無効にするために一番地が無駄になるのを防ぐものです．**PBN** や **PBF** は実行時のオブジェクト・コード上ではすべて元の値（ポート番地，ビット番号，メモリ・アドレス）に復元されている点に注意してください．**PBN** の値はシンボル・テーブル上には図 3.4 のフォーマットでリストされています．

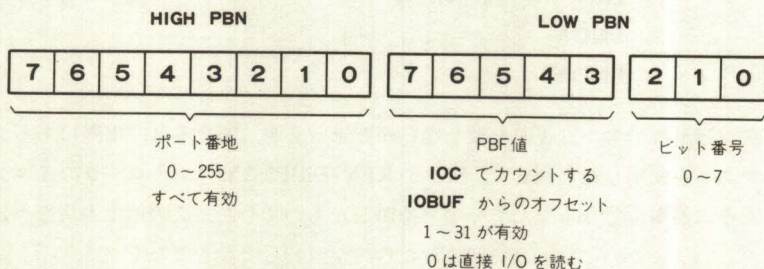
〔**例題⑱**〕 図 3.3 では RAM の番地のオフセット値を **IOC** というアセンブル変数でカウントしているが，このカウンタは最初に 0 にしなければならない．カウンタを使用せずに **IOC** と同じ値を作るにはどうすればよいか．

〔**解答例**〕 **$－IOBUF＋1** を用いる．**IOC** という変数は不要になる．

〈図3.3〉 RAM対応**PBN**の使用例

```
                                .Z80
                        PBNDEF  MACRO   PORT,BNAME,BF
                                IFNB    <BF>
                        IOC     DEFL    IOC+1
                        TEMP    DEFL    PORT*256+IOC*8
                                DS      1
                                ELSE
                        TEMP    DEFL    PORT*256
                                ENDIF
                                IRP     Q,<BNAME>
                                IFNB    <Q>
                        Q       EQU     TEMP
                                ENDIF
                        TEMP    DEFL    TEMP+1
                                ENDM
                                ENDM
                        ;
                        ON      MACRO   PBN
                                IF      (PBN AND 248) EQ 0
                                IN      A,(HIGH PBN)
                                SET     LOW PBN,A
                                OUT     (HIGH PBN),A
                                ELSE
                                LD      A,((PBN AND 248)/8+IOBUF-1)
                                SET     7 AND PBN,A
                                LD      ((PBN AND 248)/8+IOBUF-1),A
                                ENDIF
                                ENDM
                        ;
0000'                           ASEG
0000                    IOC     DEFL    0
                                ORG     8000H
8000                    IOBUF:
                        PBNDEF 3,<ONE,TWO,THREE>,B
8000                +           DS      1
                        PBNDEF 5,<RED,GREEN,BLUE>
                        PBNDEF 15,<TEN,TWENTY>,B
8001                +           DS      1
                        ;
8002                            CSEG
                                ON      RED
0000'   DB 05       +           IN      A,(HIGH RED)
0002'   CB C7       +           SET     LOW RED,A
0004'   D3 05       +           OUT     (HIGH RED),A
                                ON      TWO
0006'   3A 8000     +           LD      A,((TWO AND 248)/8+IOBUF-1)
0009'   CB CF       +           SET     7 AND TWO,A
000B'   32 8000     +           LD      ((TWO AND 248)/8+IOBUF-1),A
000E'   D3 03       +           OUT     (HIGH TWO),A
                                ON      TWENTY
0010'   3A 8001     +           LD      A,((TWENTY AND 248)/8+IOBUF-1)
0013'   CB CF       +           SET     7 AND TWENTY,A
0015'   32 8001     +           LD      ((TWENTY AND 248)/8+IOBUF-1),A
0018'   D3 0F       +           OUT     (HIGH TWENTY),A
                                END
Macros:
ON                  PBNDEF

Symbols:
0502    BLUE            0501    GREEN           8000    IOBUF
0002    IOC             0308    ONE             0500    RED
0F12    TEMP            0F10    TEN             030A    THREE
0F11    TWENTY          0309    TWO
```

図3.3では**ON**のマクロしか定義していませんが，当然**OFF**も同様に定義できます．また，1行で何ビットかをまとめて**ON**できるような図2.28の方式も使えます．**PBNDEF**や**ON**のマクロ定義はずい分変わりましたが，呼び出し方法がほとんど(**ON**の方は全く)変わっていない点も重要です．

〈図3.4〉 RAM対応PBN値のビット構成

3.2　マルチ入出力で周辺LSIをイニシャライズ

周辺用LSIにも高度な機能を持ったものが作られています．その多くは内部に持っているコントロール・レジスタやステータス・レジスタを同じ番地に重ねています．重ねる理由はアドレス線を多く使用するとピン数が増える(か逆に機能を減らす)ことになり，I/Oマップも多くを占有されてしまうからです．

レジスタを重ねて同一の番地にした結果，連続して同じ番地に数バイトを出力したり，レジスタ番号を書き込んでからその内容を読み取る必要が出てきます．通常は，

```
LD    A, 80H
OUT   (SIOAC), A
LD    A, ……
OUT   (……), A
```

の繰り返しが続きます．これを同一のポートに対する出力を1行で表現できるマクロを使用すればスッキリして見やすくなります．

3.2.1　多バイト連続出力用マクロ

このマクロの呼び出し形式は第1パラメータに出力したいポート番地を，第2パラメー

タに出力したいデータの集合を与えることにします.

```
MOUT        MACRO    PN, DATA
            IRP      Q, <DATA>
            LD       A, Q
            OUT      (PN), A
            ENDM
            ENDM
```

このマクロは単純ですから説明を要しないかと思います. 図3.5は, 実際にあるシステムでこのマクロを使用した実例です. SIOやKEY-DISPLAYコントローラのイニシャライズがこんなに簡単にできるなんて……と感激したものでした. この例で大切な点は, マクロ呼び出しのデータの中に( )で囲まれたものが混じっていることです.

マクロ **MOUT** の中の **IRP** の仮パラメータ **Q** が **LD A** の第2オペランドですから( )が出て来てもその通り, 意図通りに展開されます. すなわち, 44行目の**(BAURAT)**はイニシャライズ・データをイミディエイトでなくメモリ・アドレス **BAURAT** から取り込むことを意味しています. サブルーチン方式では同一書式の中にデータを渡したりアドレスを渡したりすることは不可能です. マクロ表記の柔軟性をよく示しています.

3.2.2 読み取るレジスタ番号を出力するマクロ

多くのステータス・レジスタの中のひとつを指定するのに, まずレジスタ番号を出力してから読み取る方式のLSI(Z80 SIOなど)を使用する時に使うマクロです.

```
            MINP     SIOC, 5
```

のように呼び出します. その定義は,

```
MINP        MACRO    PORT, REG
            LD       A, REG
            OUT      (PORT), A
            IN       A, (PORT)
            ENDM
```

となります.

〈図3.5〉 多バイト連続出力用マクロ **MOUT** の使用例

```
  1:    IODEF  0E0H,<CTC0,CTC1,CTC2,CTC3>
  2:    IODEF  0E4H,<SIOD,SIOC,SIOB,SIOV>
  3:    IODEF  0E8H,WATC
  4:    IODEF  0ECH,<NUI,NUO,DOG,NCUM,KBD,KBC>
  5:    IODEF  0F4H,<RECD,RECC,RECS,RECM,TRIGPO>
  6:   ;
  7:   ;**** CONSOLE KEY DEFF
  8:    IODEF  10,<KA,KB,KC,KD,KE,KF>
  9:    IODEF  27,<KW,KI,KN,COMMA,KRET>

 43:   ;****   PORT INITIALIZE
 44:           MOUT     CTC0,<VECCTC,7,(BAURAT)>
 45:           MOUT     CTC1,<39,96>
 46:           MOUT     CTC3,<135,(RECRAT)>
 47:           MOUT     SIOV,<2,VECSIO>
 48:           MOUT     SIOC,<1,0,4,32,6,0,7,126>
 49:           MOUT     KBC,<8,57,64,96,90H,160,0D3H,-1>
 50:           MOUT     NCUM,130 ;C,A:OUT B:IN MODE 0
 51:           MOUT     NUO,0    ;S-BY
 52:           MOUT     DOG,<3,0> ;S-BY OF T-HOLD
 53:           MOUT     RECM,0BH ;A,B OUT M0DE 1 C4,5 IN
 54:           MOUT     RECC,0
 55:           MOUT     RECD,32
```

このマクロは正常に働きますが，レジスタ0を読む時には0を出力する必要がないのに **LD** と **OUT** を消すことができません．

〔例題⑲〕 **MINP** をレジスタ0が指定された時には **LD** と **OUT** が作られないように変更せよ．

〔解答例〕 **LD** と **OUT** を **IF** と **ENDIF** で囲む．

```
IF        REG  NE  0
LD        A, REG
OUT       (PORT), A
ENDIF
IN
     ⋮
```

3.3 マクロ定義もマクロで！

RAMのエリヤ確保とシンボル定義のためのマクロを2.9.2節で定義しています．図2.27ではこのための4つのマクロを定義する例を示しました．ところで，この4つのマクロはよく似ています．**DS1**が2～4に変化するだけでその他は全く同じです．

同じ繰り返しを簡単に表現するのがマクロの機能であれば，マクロ定義をする時にもその機能が使えてもよいはずです．これはマクロ定義のネスティングです．通常はネスティングは呼び出し段階で使用しますが，このようにマクロ定義でのネスティングはあまり例がありません．ただ，マクロ機能を自在に操る上ではマクロ定義のネスティングもよく理解しておかなければなりません．

図3.6は図2.27と同じ4つのマクロ定義を，マクロ定義用のマクロ**QQQ**を4回呼び出すことで実現しています．その4回は**IRP**でマクロ名を4つ与える方法で起動します．4行目の**MACRO**は**QQQ**をマクロとして定義するためのものですが，5行目の**MACRO**はこの時点では**MACRO**という文字のならびにしかすぎません．4行目の**MACRO**が，5行目の**JJ**から9行目の**ENDM**までをマクロの内容として定義し，10行目の**ENDM**で**QQQ**が完結します．

〈図3.6〉マクロ定義でネスティングを用いた例

```
 1:             .Z80
 2:             .LALL
 3:  Z1         DEFL    0
 4:  QQQ        MACRO   JJ,LL
 5:  JJ         MACRO   KK
 6:             IRP     XX,<KK>
 7:  XX:        DS      LL
 8:             ENDM
 9:             ENDM
10:             ENDM
11:             IRP     K,<BYTE,WORD,TRIP,QUAD>
12:  Z1         DEFL    Z1+1
13:             QQQ     K,%Z1
14:             ENDM
15:
16:             BYTE    <B1,B2,B3,B4>
17:             WORD    <ADRS1,ADRS2,ADRS3>
18:             BYTE    <Y1,Y2,Y3>
19:             TRIP    <DATA,DATB,DATC,DATY,DATZ>
20:             QUAD    <PAR1,PAR2>
21:             WORD    <POINT1,POINT3>
22:  DATEND     EQU     $
23:             END
```

マクロ定義の段階で，**ENDM**には番号も名前もつけてなくても**ENDM**の相棒が現れた数だけ無視され，自分の相棒が現れた時にはじめて定義が完了するようになっています．

13行目で**QQQ**が呼び出された時には，その場所に5行目から9行目までの文字のならびが展開されます．しかも，それが**IRP**の中からですから**IRP**のパラメータの数だけ展開されることになります．この展開のときに，はじめて5行目にあった**MACRO**という文字列がマクロ定義の命令として生きます．このようにして**IRP**で与えた4つのパラメータ**BYTE**～**QUAD**を名前として持つマクロが定義されます．図3.7ではこの展開過程がよくわかります．

よく似たマクロをいくつか定義したい時は，通常はマクロ呼び出しのネスティングを使用します．マクロ定義をネスティングするメリットはどこにあるでしょうか．それはマクロ展開時間が短くなることと，書く行数が少なくてすむことの二点です．呼び出しでのネスティングでは，4つのマクロ名を**MACRO**の行に書かないわけには行きません．これを書けば中身と**ENDM**と合わせてひとつ当たり3行で合計12行が必要です．それと，その各マクロから呼び出される共通のマクロと合計5つのマクロを直接定義しなければなりません．図3.6では12行でマクロ定義をすべて完了しています．

スピードについては，呼び出しのネスティングでは，毎回のマクロ呼び出しに際してさらにその中から別のマクロをさがしますから当然時間がかさみます．マクロ定義において複雑な動きをしても，定義は1回だけであり，トータルの時間では呼び出し時の方が効いてきます．

この例ではデータ・エリヤに**DS**を作るだけですから，マクロ定義を保持するためのメモリ（アセンブラが使用する作業用メモリ）は少なく問題になりませんが，もし内容が90行似ていて10行が違っているマクロを10種類くらい定義する時は，作業エリアを大きく取るために，呼び出しネスティングにするべきです．アセンブラは作業用メモリが少ないと，仮に足りなくなるようなことがなくてもスピードが下がります．

〔**例題⑳**〕　図3.6で，5バイト定義用マクロ**QUIN**と6バイト定義用の**HEXA**を追加するには，どこをどのように変えればよいか．

〔**解答例**〕　11行目の**QUAD**のあとに，

QUAD, QUIN, HEXA>

を追加すればよい．バイト数は**Z1**がカウントされて自動的に5バイトと6バイトのエリアが定義される．

〈図3.7〉 図3.6の展開過程

```
                                .Z80
                                .LALL
 0000                   Z1      DEFL    0
                        QQQ     MACRO   JJ,LL
                        JJ      MACRO   KK
                                IRP     XX,<KK>
                        XX:     DS      LL
                                ENDM
                                ENDM
                                ENDM
                                IRP     K,<BYTE,WORD,TRIP,QUAD>
                        Z1      DEFL    Z1+1
                                QQQ     K,%Z1
                                ENDM
 0001           +       Z1      DEFL    Z1+1
                +               QQQ     BYTE,%Z1
                +       BYTE    MACRO   KK
                +               IRP     XX,<KK>
                +       XX:     DS      1
                +               ENDM
                +               ENDM
 0002           +       Z1      DEFL    Z1+1
                +               QQQ     WORD,%Z1
                +       WORD    MACRO   KK
                +               IRP     XX,<KK>
                +       XX:     DS      2
                +               ENDM
                +               ENDM
 0003           +       Z1      DEFL    Z1+1
                +               QQQ     TRIP,%Z1
                +       TRIP    MACRO   KK
                +               IRP     XX,<KK>
                +       XX:     DS      3
                +               ENDM
                +               ENDM
 0004           +       Z1      DEFL    Z1+1
                +               QQQ     QUAD,%Z1
                +       QUAD    MACRO   KK
                +               IRP     XX,<KK>
                +       XX:     DS      4
                +               ENDM
                +               ENDM

                                BYTE    <B1,B2,B3,B4>
                +               IRP     XX,<B1,B2,B3,B4>
                +       XX:     DS      1
                +               ENDM
 0000'          +       B1:     DS      1
 0001'          +       B2:     DS      1
 0002'          +       B3:     DS      1
```

```
0003'          +      B4:     DS      1
                              WORD    <ADRS1,ADRS2,ADRS3>
               +              IRP     XX,<ADRS1,ADRS2,ADRS3>
               +      XX:     DS      2
               +              ENDM
0004'          +      ADRS1:  DS      2
0006'          +      ADRS2:  DS      2
0008'          +      ADRS3:  DS      2
                              BYTE    <Y1,Y2,Y3>
               +              IRP     XX,<Y1,Y2,Y3>
               +      XX:     DS      1
               +              ENDM
000A'          +      Y1:     DS      1
000B'          +      Y2:     DS      1
000C'          +      Y3:     DS      1
                              TRIP    <DATA,DATB,DATC,DATY,DATZ>
               +              IRP     XX,<DATA,DATB,DATC,DATY,DATZ>
               +      XX:     DS      3
               +              ENDM
000D'          +      DATA:   DS      3
0010'          +      DATB:   DS      3
0013'          +      DATC:   DS      3
0016'          +      DATY:   DS      3
0019'          +      DATZ:   DS      3
                              QUAD    <PAR1,PAR2>
               +              IRP     XX,<PAR1,PAR2>
               +      XX:     DS      4
               +              ENDM
001C'          +      PAR1:   DS      4
0020'          +      PAR2:   DS      4
                              WORD    <POINT1,POINT3>
               +              IRP     XX,<POINT1,POINT3>
               +      XX:     DS      2
               +              ENDM
0024'          +      POINT1: DS      2
0026'          +      POINT3: DS      2
0028'                 DATEND  EQU     $
                              END
Macros:
BYTE            QQQ             QUAD            TRIP            WORD

Symbols:
0004'   ADRS1           0006'   ADRS2           0008'   ADRS3
0000'   B1              0001'   B2              0002'   B3
0003'   B4              000D'   DATA            0010'   DATB
0013'   DATC            0028'   DATEND          0016'   DATY
0019'   DATZ            001C'   PAR1            0020'   PAR2
0024'   POINT1          0026'   POINT3          000A'   Y1
000B'   Y2              000C'   Y3              0004    Z1
```

3.4　パラメータの省略時解釈

2.9.3節で**PBN**の利用法として**WHEN**というマクロを定義しています．そして宿題として第2パラメータの**Z**または**NZ**を**ON/OFF**と書けるように定義するようにとあります．これをやってみましょう．それには，**IN**と**BIT**の2行はそのままで**JP**の1行を，

```
IFIDN   <POL>, <ON>
JP      NZ, ADS
ELSE
JP      Z, ADS
ENDIF
```

とこのように変更すればよいのですが，おかしなことにここには**OFF**という文字は全く出てきません．それでも

```
WHEN    LOCK, OFF, SHORI 1
```

として呼び出せば確かに**JP　Z, …**が作られますから宿題の題意にはかなっています．それでは，

```
WHEN    LOCK, , SHORI 1
```

として第2パラメータを空にしたらどうでしょうか．これでも実は**OFF**と同じ結果になります．当然，

```
WHEN    LOCK,?, ……
```

と書いてもやはり**OFF**と同じです．つまりは**ON**以外はすべて**OFF**と同じです．

ところで第3パラメータを省略してしまったらどうなるでしょうか．

```
WHEN    LOCK, ON
```

これでは**JP　NZ,**となってエラーになります．第3パラメータを省略した時には，このマクロ命令でループしてくれると便利です．しかし，

〈図 3.8〉 パラメータを省略したときの解釈例

```
                                                .Z80
                                        WHEN    MACRO   PBN,POL,ADS
                                                IN      A,(HIGH PBN)
                                                BIT     LOW PBN,A
                                                IFB     <ADS>
                                                IFIDN   <POL>,<ON>
                                                JP      NZ,$-4
                                                ELSE
                                                JP      Z,$-4
                                                ENDIF
                                                ELSE
                                                IFIDN   <POL>,<ON>
                                                JP      NZ,ADS
                                                ELSE
                                                JP      Z,ADS
                                                ENDIF
                                                ENDIF
                                                ENDM
                                        ;
0503                                    LOCK    EQU     503H
0000'                                   STT:    WHEN    LOCK,ON,PPP
0000'   DB 05           +                       IN      A,(HIGH LOCK)
0002'   CB 5F           +                       BIT     LOW LOCK,A
0004'   C2 0008'        +                       JP      NZ,PPP
0007'   00                                      NOP
0008'                                   PPP:    WHEN    LOCK
0008'   DB 05           +                       IN      A,(HIGH LOCK)
000A'   CB 5F           +                       BIT     LOW LOCK,A
000C'   CA 0008'        +                       JP      Z,$-4
                                                WHEN    LOCK,ON
000F'   DB 05           +                       IN      A,(HIGH LOCK)
0011'   CB 5F           +                       BIT     LOW LOCK,A
0013'   C2 000F'        +                       JP      NZ,$-4
                                                WHEN    LOCK,,STT
0016'   DB 05           +                       IN      A,(HIGH LOCK)
0018'   CB 5F           +                       BIT     LOW LOCK,A
001A'   CA 0000'        +                       JP      Z,STT
                                        ;
                                                END
```

〈図 3.9〉 **DJNZ** を使った **WAIT** におけるループ回数と時間の関係

タイム値	実行クロック数	実行時間 μs	カバー範囲 μs
0	3330	1332.0	0〜2
1	15	6.0	3〜8
2	28	11.2	9〜13
3	41	16.4	14〜18
4	54	21.6	19〜24
5	67	26.8	25〜29
255	3317	1326.8	1325〜1329

(クロック 2.5MHz のとき)

LD B, タイム値
DJNZ $
最も簡単な時間待ち

WHEN LOCK, ON, $

と書くと，このマクロ命令でループするのではなく，**JP NZ, $**となって **JP** だけでループしますから永久に抜け出られなくなってしまいます．マクロの中身を知っていれば，これは当然**$−4** でなければならないことがわかります．

そこで，**WHEN** を省略した時，自己ループになるように定義して呼び出した例が図 3-8のようになります．この定義では，

WHEN LOCK

は，**LOCK** が(だまっていれば **OFF** だから) **OFF** なら，このマクロを再び実行せよ……という意味です．言い換えれば **LOCK** が **ON** になるまで待てというように取れます．

3.4.1 時間待ち用マクロ

最も簡単な時間待ちルーチンは **DJNZ** で作るものです．**DJNZ** が自己ループに入る前に，レジスタＢにロードした値によってループ終了までの時間(クロック数)が決まります．クロックを数える時間待ちは DMA やインタラプトを使用しているシステムでは正確には出せませんが，何となくこの程度の時間が取れればよいという用途には重宝します．この時間待ちは，Ｂに置く値によって 6 〜1332 μs の時間が作れます．この関係を図 3.9 に示します．この表のクロック数は **LD B** の分も含まれています．これをそのままマクロに

して，

```
WAIT    MACRO   TIME
        LD      B, TIME
        DJNZ    $
        ENDM
```

WAIT　5と呼び出すと26.8 μs後に次の命令に進みます．しかし，これでは待ちたい時間とパラメータの値の関係を意識しながら使わなければなりません．できればμsを表す数値で書きたいところです．

そこで，アセンブラの計算機能を利用して，**LD　B**を

```
LD    B, 0+(5 * TIME+11)/26
```

と変更します．この式は図3.9のカバー範囲の値を書いた時，タイム値を計算してくれます．すなわち，**WAIT　10**と書けば**LD　B，2**となります．

DJNZだけでは，最大時間が約1.3 msで実用上は短すぎます．そこで，ms単位での時間待ちをマクロ定義しておきたいのですが，同じ定義をするなら単位指定をできるようにした方が便利です．ついでに，使用頻度の高い単位は省略でき，使用頻度の高い時間はその数値も省略できるようにします．

通信制御ではμs単位で，メカ制御はms単位でというシステムで，単位指定や省略指定のマクロは非常に便利です．図3.10がその定義および呼び出し展開例です．マクロ定義を変更しなくても，**CONSTT**に**EQU**する値を変えれば，使用するプログラムごとに一定時間の値を変更できます．変更が不要ならマクロの中で数値を書いてしまった方が使いやすくなります．

〔例題㉑〕　図3.10のマクロ定義を変更して，時間の単位に**SEC**を使えるようにせよ．ただし，**SEC**単位のときは，最大設定値は**65**(すなわち65秒)まででよい．

〔解答例〕　図3.11に示す．外のカウンタにBCを使用し，最大65536 msまで設定可能にし，その値を**SEC**単位から1000倍して与えている．

WAIT　　50000　と

WAIT　　50，SEC　とは

全く同じコードに展開される．

〈図3.10〉 時間待ちのためのマクロ **WAIT**

```
                                .Z80
                        WAIT    MACRO   TIME,UNIT
                                IFIDN   <UNIT>,<USEC>
                                LD      B,0+(5*TIME+11)/26
                                DJNZ    $
                                ELSE
                                IFB     <TIME>
                                LD      C,CONSTT
                                ELSE
                                LD      C,TIME
                                ENDIF
                                LD      B,191
                                DJNZ    $
                                DEC     C
                                JR      NZ,$-5
                                ENDIF
                                ENDM
                        ;
000A                    CONSTT  EQU     10
                                WAIT
0000'   0E 0A       +           LD      C,CONSTT
0002'   06 BF       +           LD      B,191
0004'   10 FE       +           DJNZ    $
0006'   0D          +           DEC     C
0007'   20 F9       +           JR      NZ,$-5
                                WAIT    300,USEC
0009'   06 3A       +           LD      B,0+(5*300+11)/26
000B'   10 FE       +           DJNZ    $
                                WAIT    5,MSEC
000D'   0E 05       +           LD      C,5
000F'   06 BF       +           LD      B,191
0011'   10 FE       +           DJNZ    $
0013'   0D          +           DEC     C
0014'   20 F9       +           JR      NZ,$-5
                                WAIT    100
0016'   0E 64       +           LD      C,100
0018'   06 BF       +           LD      B,191
001A'   10 FE       +           DJNZ    $
001C'   0D          +           DEC     C
001D'   20 F9       +           JR      NZ,$-5

                                END
```

〔**宿題**〕 17行目に**WAIT**で自己呼び出しとなっている．どのような条件でネスティングが終結するか？

〈図 3.11〉 例題㉑の解答

```
 1:              .Z80
 2:   WAIT       MACRO    TIME,UNIT
 3:              IFIDN    <UNIT>,<USEC>
 4:              LD       B,0+(5*TIME+11)/26
 5:              DJNZ     $
 6:              ELSE
 7:              IFIDN    <UNIT>,<SEC>
 8:              LD       BC,TIME*1000
 9:              ELSE
10:              IFB      <TIME>
11:              LD       BC,CONSTT
12:              ELSE
13:              LD       BC,TIME
14:              ENDIF
15:              ENDIF
16:              PUSH     BC
17:              WAIT     994,USEC
18:              POP      BC
19:              DEC      BC
20:              JR       NZ,$-7
21:              ENDIF
22:              ENDM
```

3.5　マクロ内部で定義，参照されるマクロ名は LOCAL にできる

大きなプログラムをアセンブラ・レベルで記述するときは，シンボル名がダブらないようにするのに苦労します．このことはマクロ名についても当てはまります．通常のシンボルについては，マクロ内で参照されるラベルをローカル化することで，一般的な名称とダブる可能性は減少します．マクロ名についても，システム上で常用マクロのファイルが別になっていて，さらに外からは見えない部分でマクロのネスティングが行われている場合，どのような名前がつけられているのか，作った本人でも時間が経つと忘れがちです．

頭文字を限定したり，文字数を変えてみたりして重複を避けるのが決まり手ですが，煩わしいものではあります．そこで，特定のマクロ内でしか呼び出さないマクロは，その中で定義して **LOCAL** 宣言が適用できないかとやってみたところ，マクロ名についても正しく動作しました．マクロ名を **LOCAL** 指定すると，そのマクロのソース・プログラム上の名前は，そのマクロ内でのみ重複しないようにすればよく，名前の苦労は激減します．

LOCAL 化されたマクロの名前は，他のシンボルの場合と同じく，**..0000** から始まる下 4 桁が 16 進の文字になっている記号になります．マクロ名とその他のシンボルは通し番

号になり，区別はされません．結果として，どのような記号がマクロ名につけられたかについては，リストのMacrosの欄に出力されるので確認はできます．図3.12にマクロを**LOCAL**にした例を示します．

3.6 マクロ内からサブルーチンを呼ぶ

これまでのマクロ定義は，単独のものとマクロ内でマクロ呼び出しを行うもの，そしてマクロ内でマクロ定義を行うものなどがありましたが，生成される機械語コードにはマクロ定義されたもの以外に飛び出すものはありませんでした．これは例として挙げているものが簡単な処理しか行っていないからです．ところが，実際のプログラムではかなり長い処理もマクロ呼び出しでパラメータによる柔軟な対応能力を持たせたくなります．

このようなとき，その長い処理をすべてマクロの中で定義していたのでは毎回毎回大量のプログラム・メモリを費やしていくことになります．これを避けるには処理のうちパラメータを変えても変化しない部分をサブルーチン化して，マクロの外に書いておき，マクロ展開時には変更部分だけ作り出して，共通部分はサブルーチンの**CALL**命令にしてしまいます．また，処理内容が全く異なるものをひとつのマクロで扱いたいならば，パラメータによって呼び出すサブルーチンを変更する方法も使えます．

〈図3.12〉マクロ名の**LOCAL**定義

```
                              .LALL
                   TEST       MACRO
                              LOCAL LMACRO      ;ﾅｲﾌﾞ ﾏｸﾛ
                   LMACRO     MACRO
                              DB          'ABC'
                              ENDM
                              LMACRO
                              RET
                              ENDM
                   ;
                              TEST
              +    ..0000     MACRO
              +               DB          'ABC'
                              ENDM
              +               ..0000
0000'   41 42 43   +          DB          'ABC'
0003'   C9         +          RET
                              END
Macros:
..0000        TEST
```

図3.13はマクロ定義の中に**CALL**を使っている例です．このサブルーチンは2バイトしかありませんが，この16ビット加算がもし乗算であってもマクロ展開あたりのバイト数は全く変わらない点に注意してください．もちろん乗算であればサブルーチンは大きくなりますが，サブルーチンがマクロ定義からはずされていますからマクロ呼び出しにかかわらず1度だけ書かれているわけで，展開の都度増えていく分を節約できます．

このマクロの中身は**LD, LD, CALL, LD**という形式になっていますが，これはマクロを使わないで多くのサブルーチン処理を呼び出すときの典型的パターンです．この典型的パターンをそのままマクロ化したのが図3.13であり，書式の上では見やすく一行に納まっています．

マクロ内で**CALL**を使用するのは必ずしも大きなルーチンのメモリ節約ばかりではありません．数バイトのデータ・ブロックを扱うときに**INC　HL**が連続して何個かずつ使用されるとき，その頻度が高ければたとえ5バイトでも**CALL**で3バイトにすれば10～20％のエリヤ節約になるケースもあります．このようなとき，

```
        INC     HL  ┐
         ⋮          │ 必要な数までならべる
        INC     HL  ┘
INCHL:  RET
```

〈図3.13〉マクロ定義に**CALL**を使った例

```
                                .Z80
                        ADDW    MACRO   P1,P2,P3
                                LD      HL,(P1)
                                LD      DE,(P2)
                                CALL    ADDW
                                LD      (P3),HL
                                ENDM
0000'   19              ADDW:   ADD     HL,DE
0001'   C9                      RET

                                ADDW    AAA,BBB,CCC
0002'   2A 000F'        +       LD      HL,(AAA)
0005'   ED 5B 0011'     +       LD      DE,(BBB)
0009'   CD 0000'        +       CALL    ADDW
000C'   22 0013'        +       LD      (CCC),HL
000F'                   AAA:    DS      2
0011'                   BBB:    DS      2
0013'                   CCC:    DS      2
                                END
Macros:
ADDW
```

という終わりに名前がついたサブルーチンを用意しておいて，HL を **n** だけ **INC** したいとき，

```
CALL    INCHL-n
```

として呼び出せば目的は達せられます．

しかし，よく考えてみれば3回までは直接 **INC HL** を書いた方が速くなりますから，その判断をプログラマが行わなければなりません．そこで，

```
INCHL   MACRO   CNT
        IF   CNT  GT  3
        CALL    INCHL-CNT
        ELSE
        REPT    CNT
        INC     HL
        ENDM
        ENDIF
        ENDM
```

とマクロ化します．これで HL を何個インクリメントしても3バイトしか食われません．サブルーチン自身も，**INC　HL** をならべなくても **REPT** で必要数作れば手数はかかりません．

この方法では，**INCHL　10** 程度まではよいとして，**INCHL　100** を使うためにはサブルーチンも100個の **INC　HL** を並べなければなりません．これではメモリ・エリアも無駄なばかりでなく，実行時間もかかります．そこで **INCHL　8** 以上は **ADD　HL** で計算する別のサブルーチンにしてみます．

```
        IF  CNT  GT  7
        LD     BC, CNT
        CALL    INCHL8
        ELSE
          ⋮
```

ところがこれでは **CALL** にする価値はなくなり，直接 **ADD　HL, BC** の方が4バイト

〈図3.14〉16ビット・レジスタの **INC** と **DEC**

レジスタ＼動作	HL	BC	DE	IX	IY	単位
INC	42	50	46	46	46	クロック
DEC	50	66	58	110	110	クロック
INC	6	8	8	7	7	バイト
DEC	8	12	10	16	16	バイト

(a) 加算，減算による **Push pop** 含む

(命令構成によっては改善の余地はある)

INC・**DEC** の数	クロック数	
	HL,BC,DE	IX,IY
1	6	10
2	12	20
3	18	30
4	24	40
5	30	50
6	36	60
7	48	70
8	54	80
9	60	90
10	66	100
11	72	110

(b) **INC** または **DEC** による

で済み，しかも速くなります．4バイト必要になるのはまだよいとしても，BCもDEも壊せない時は **PUSH** と **POP** で合計6バイト必要です．こうなると手詰まりで，マクロ機能上はこれが限度です．

〔例題㉒〕 実行速度の関係から，**INCHL 8** 以上を別サブルーチンにするように考えた．もし **DECHL** というマクロを定義するとすれば，やはり8以上から別ルーチンでよいか．

〔解答例〕 だめ．**DECHL n** を計算するのには **SBC** を使わなければならず，その前にキャリを消す必要もあるので9以上とした方がよい．

3.7 一度だけ展開される部分を含むマクロ――再展開防止法

前節では，マクロ展開中に **CALL** が使用される例について説明しました．この方法が長い処理において有利なことは確かですが，唯ひとつ困るのは同一目的のためにマクロ定義とサブルーチンをペアで管理しなければならないことです．もしマクロ定義とサブルーチンをひとつのファイル中に入れておいて，メイン・プログラムにインクルードしたとすれば，すべてのサブルーチンがオブジェクト・コードとして作られてしまいます．

マクロとサブルーチンが多くなってくると，インクルードした中に不要のサブルーチンが入らないようにファイルを分けたり，組み合わせを変えるなどの手間がかかります．す

〈図3.15〉 自マクロの再定義

```
                                        .LALL
                                        .Z80
                                LD      MACRO   PP,QQ
                                        INC     B
                                LD      MACRO   RR,SS
                                        ADD     RR,SS
                                        ENDM
                                        ENDM

                                        LD      A,0
   0000'    04              +           INC     B
                            +   LD      MACRO   RR,SS
                            +           ADD     RR,SS
                                        ENDM
                                        LD      A,0
   0001'    C6 00           +           ADD     A,0
                                        END
```

べてのサブルーチンを取り入れてもメモリに余裕があれば実害はありませんが，実際には許されないことが多いでしょう．

そこで，サブルーチンを扱うマクロでは定石的な手段が用いられます．それはサブルーチンも含めてマクロの中に定義しておき，そのうちの毎回展開する必要のない部分を1度だけしか展開しない方法です．マクロ定義は大きくなりますが，呼び出されたマクロから呼ばれるサブルーチンだけがアセンブルされることになり，メモリの無駄がありません．また，ひとつのマクロ定義だけを管理すればよくなり，無用なトラブルや手違いが減ります．

マクロを一度だけ展開させるには，マクロ定義のときにそうなるように細工が必要です．それにはいくつかの方法が用いられています．

3.7.1　自マクロ再定義による方法

3.3節ではいくつかのマクロ定義を行うためのマクロを説明しました．そのマクロは別のマクロを定義したわけですが，こんどはマクロ定義の中に，そのマクロ自身を定義する **MACRO** が書かれているもので，自マクロ再定義と呼んでいます．自マクロだから再定義つまり定義し直すことになるのです．要は，最初に呼び出された時にはサブルーチン全体と，呼び出し部分，パラメータ(ここでは実行時にレジスタに乗せたりするものを指す)設定と受け取り部分を展開し，その展開の中で次回以後はサブルーチンそのものの展開を含

んでいない形に定義し直すわけです．

図3.15を見てください．ここでは**LD**という名前で，パラメータ2個を受け取るマクロがまず定義されています．その中で再び**LD**を定義する部分が入っていて，展開時にこの定義が有効になります．再定義された結果，マクロ**LD**は**ADD**の1行だけとなり，2回目以降は何度展開されても展開されるのは**ADD**の1行です．**INC　B**は2回目以降永久に展開されません．

そこで，**INC　B**の部分にサブルーチン呼び出しとサブルーチンそのものを書いておき，**ADD**の部分にサブルーチン呼び出し部分のみを書いておけばサブルーチン再展開がされないわけです．

ところで，再定義されたマクロの古い方の中身はどうなるのでしょうか．アセンブラによっては古いものの上に新しいものを積み重ねていくものもありますが，M80では古いものは残りません．すなわち同じマクロを何万回も定義し直しても，それが原因でメモリ・エリアが足りなくなることはありません．

3.7.2 定義済みフラグによる方法

フラグによる方法はマクロの内容を定義し直さず，展開ずみのフラグをプログラムの先頭ですべて(いくつか使用される場合)0にしておき，マクロ内で最初の展開であるか，2回目以降であるか区別できるようにマクロ展開時にフラグを書き換えるものです．

```
FLAG1     DEFL   0      (プログラムの先頭)
            :
××××      MACRO
       (毎回必要な部分)
          IF      FLAG1  EQ  0
FLAG1     DEFL   1
       (最初のみ必要な部分)
          ENDIF
          ENDM
```

これはわかりやすい方法ですが，難点が2つあります．ひとつは**FLAG**のイニシャライズです．この数が増えてくると，そのためにマクロを使うようです．そして**FLAG**に名前をつけなければならず，またまた重複しないマクロ名ゆかりの名前をつけるのに苦労しま

す．

また，シンボル・テーブルも占有されるなど，要素が増えていないのに名前が増えるのはどうも面白くありません．

〔例題㉓〕 再定義による場合，単純な方法では毎回使用する部分を2度（最初の定義と再定義）書かなければならない．これを1度の定義で済ませる方法は？

〔解答例〕 再定義と自己呼び出しを並用する．

```
××××    MACRO
        (初回使用)
××××    MACRO   ……(再定義)
        (毎回使用)
        ENDM
        ××××    ……(再定義されたものを呼び出す)
        ENDM
```

〔例題㉔〕 フラグによる場合と再定義による場合とどちらが多くのマクロを扱えるか（使用するメモリ量が少ない方がよい）

〔解答例〕 再定義の方．再定義された結果，マクロは毎回使用する部分のみになり，より多くの他のエリアが残っている．フラグによる方法では条件判断によってアセンブル対象からはずしているだけで，マクロそのものは全体をかかえている（M80以外のアセンブラでは，定義されたマクロを保持する方法がさまざまで，再定義にあたっても古い方を消さないものでは当然再定義方式は不利になる）．

3.8 マクロ名の制限は――シンボルとの重複，命令コードとの重複

一般的には，マクロはマクロ命令としての扱いですから，データのシンボルと一致するケースは少ないと思われますが，フラグ用のアセンブル変数，カウンタ用のアセンブル変数，マクロ内から **CALL** するサブルーチンの入口ラベル，そしてアセンブラ組み込みの疑似命令（**.PRINTX** や **NAME** など）と機械語命令など重複しやすいものです．現に，8080コードでの疑似命令 **SET** が Z80 の機械語命令と重複しました．

やはり，命令コードは簡潔明瞭が喜ばれます．使用頻度が高いが故にマクロ化するわけですから，その名前がギクシャクすると作業性も下がります．ところで，3.7.1節では触れませんでしたが，図3.15にはすでに**LD**というマクロ名が使われています．ということはM80では機械語と同じコードのマクロ命令が作れることになります．

3.8.1 マクロ名とシンボルの重複

それでは，**LD**をマクロ命令に使ってしまった時，もとの本家**LD**はどうなるのでしょうか．これは2度と使用することはできません．その理由は，マクロは定義できても消す方法が用意されていないからです．筆者は以前から一度マクロ定義された機械語コードを元の木阿弥に戻す方法を考えていましたが，特定の命令以外は無理のようです．特に，**LD**はオペランドによるバリエーションが多すぎて再定義では不可能です．

また，図3.16によれば，何と疑似命令**IF**もマクロ名として使えるのです．しかし，**IF**も**LD**もマクロにしてしまえば元の意味としては使えなくなります．ただし，**IF**については，**IFT**と**COND**の兄弟分が全く同じ意味として解釈できますから，マクロにしてつぶしてしまっても実害はありません．3.4節の**WHEN**というマクロは**IF**と変更するともっと“**IF**”らしく見えてきます．

というわけで，通常は機械語命令や疑似命令との重複はエラーにはならないが，自主規制して使わない方が無難です．まあ，これらの命令は固定されていますから，ついうっかりマクロ名に使ってしまうというようなミスは起こらないと思います．マクロ名同士の重複は再定義と見なされてエラーにはなりません．これは十分注意しなければなりません．

さて，一般のシンボルとマクロ名の重複はどうでしょうか．多くのアセンブラではこれ

〈図3.16〉疑似命令**IF**もマクロ名として使える

```
                                  ; Even if { IF } can be a MACRO name !!
                                  ; ... BUT ORIGINAL MEANING IS LOST
                                  ;
                                          .Z80
                                  IF      MACRO   P1
                                          LD      A,(P1)
                                          ENDM

                                          IF      ABC
0000'   3A 0005           +               LD      A,(ABC)
0005                              ABC     EQU     5
                                          END
```

らの間で重複を許していません．つまり，マクロ名として定義された名前はラベルや **EQU** で定義するとエラーになります．ところが，M80 ではマクロ名とラベルや定数との重複を許しています．図 3.13 ではマクロ名とサブルーチンの入口ラベルに同じ **ADDW** を使用しています．マクロ・リストとシンボル・テーブルの両方に **ADDW** が登録されているのが確認できます．もちろん，参照する場合も文法上で判断してマクロとシンボルを混同することはありません．このおかげで，M80 ではマクロ内からサブルーチンを呼ぶ場合，単一のサブルーチンであればマクロ名と同じ名前にしておけます．このことが特にマニュアルに書いてないのは問題ですが，M80 は優れたマクロ処理機能を持っていると評価できます．

3.8.2　再展開防止用マクロ

3.7 節では自マクロ再定義とフラグ・チェックの方法について説明しました．これをさらに検討して，フラグを使用しない条件判断の方法を考えてみました．それはサブルーチンの入口名が定義されているかどうかのチェック **IFDEF** を用いる方法です．しかし，これは失敗でいろいろな条件を組み合わせたりしてみてもだめでした．

そこで，特殊な演算子 **TYPE** を使って，条件判断 **IF** と組み合わせて正しい判断ができるようになりました(**TYPE** については 2.5 節，図 2.9 参照)．その構成は，

```
××××        MACRO
           (毎回使用部分)
            IF  20H  AND  NOT  TYPE  SNAME
SNAME       EQU    $+3
            ENDIF
            IF    SNAME  EQ  $+3
           (初回のみ使用部分)
            ENDIF
```

というものです．「まず，毎回使用部分を展開した後，**SNAME**(これはサブルーチンの入口名になる)が定義されているかを **TYPE　SNAME** のビット 5 で見て，定義されていなければ **$+3** に定義します．この **EQU** はパス 1 の初回でのみ行われます．

ここで条件を閉じて，すぐ次に **SNAME** の値が **$+3** と一致するかチェックします．初回では **EQU** したばかりですから当然一致します． 2 回目以降は **EQU** されませんからそ

の時の**$+3** とは一致しません．パス 2 では **EQU** は行われません(パス 1 で **EQU** されているから)が，初回は**$+3** と一致するという事実は変わりません．したがって，パス 1 と同様の展開結果になります．

プロセスが悪いとパス 1 とパス 2 で展開結果が変わり，エラーが発生します．この方法によれば，**SNAME** としてマクロの名前と同じものを使っても正常に動作します．そこで欲を出して，この展開するかどうかのチェック部分をマクロにして 1 行で書けないものかと考えました．

3.8.3　IF □□と MACRO は交差してもよい

しかし，具合が悪いことに，チェックのための 4 行の中には **IF～ENDIF** と **IF** が含まれています．**ENDIF** はサブルーチンの終わりになるのでマクロの中に入りません．それでもやってみたらうまく行きました．マクロ内の **IF** をマクロ外の **ENDIF** で終結することが可能なのです．通常は **MACRO～ENDM** と **IF～ENDIF** は交差状態にしない方が安心ですが，このような使い方もできるとなると他にも利用法がありそうな気もしてきます．

ついでに，サブルーチンを一度だけ展開する場合，初回だけは生成されるサブルーチンを飛び越すための **JR** か **JP** が必要になるのでこれもまとめて面倒を見たのが図 3.17 です．マクロ **CHECK** はマクロの中から呼び出される展開チェック用のマクロというわけです．マクロ **TEST** を使って展開例を示してあります．この例では **DB　PA** が毎回展開の部分，**DB　PA+5** が一度のみ展開されるサブルーチン本体ということになります．

図 3.18 でもほぼ同じことを **IFDEF** を使ってチェックしていますが，チェック部分に **EXITM** が入っているのでこの部分だけ取り出してマクロ化すると正しく働きません．それは **EXITM** が，そのレベルのマクロを閉じるだけになっているからです（アセンブラによっては，**EXITM** がすべてのマクロを閉じてしまうものもありますが，M80 ではそのレベルだけです)．

〔例題㉕〕　図 3.17 ではマクロ名と同じ **TEST** を **EQU** したり **TYPE** で見たりしている．図 3.18 ではマクロ名とラベルを変えている．これを，もし同一の名前にしたらどうなるか．

〈図 3.17〉 **IF** とマクロが交差してもチェック可能なマクロ **CHECK**

```
                                  .Z80
                                  .LALL
                        CHECK     MACRO     MNAME,EM
                                  IF        20H AND NOT TYPE MNAME
                        MNAME     EQU       $+3
                                  ENDIF
                                  IF        MNAME EQ $+3
                                  JP        EM
                                  ENDM
                        TEST      MACRO     PA
                                  LOCAL     EM ←EMは毎回使うのでLOCALにする
                                  DB        PA ←毎回使用部分
             1行のみでJPを作成→   CHECK     TEST,EM
                                  DB        PA+5 ←初回のみ展開されるべき部分
                        EM:
                                  ENDIF
                                  ENDM

                                  TEST      1
0000'   01              +         DB        1
                        +         CHECK     TEST,..0000
                        +         IF        20H AND NOT TYPE TEST
                        + TEST    EQU       $+3
                        +         ENDIF
                        +         IF        TEST EQ $+3
0001'   C3 0005'        +         JP        ..0000 ←サブルーチン本体を飛び
0004'   06              +         DB        1+5          越す
0005'                   + ..0000:
                        +         ENDIF
                                  TEST      2
0005'   02              +         DB        2
                        +         CHECK     TEST,..0001
                        +         IF        20H AND NOT TYPE TEST
                        + TEST    EQU       $+3
                        +         ENDIF
                        +         IF        TEST EQ $+3
                        +         JP        ..0001
                        +         DB        2+5
                        + ..0001:
                        +         ENDIF
                                  END
```

〔解答例〕 エラーになる．**IFDEF** で見ると，マクロ定義されただけで「定義ずみ」となり，1度も **EQU** されなくなって **ADDW** を参照することができなくなる．本来ならばマクロ定義用の **IFDEFM**(たとえば)とシンボル定義用の **IFDEFS** とに区別されているべきである．

〈図 3.18〉 **IFDEF** を使った例

```
                                      .Z80
                              ADDW    MACRO   P1,P2,P3
                                      LD      HL,(P1)
                                      LD      DE,(P2)
                                      CALL    ADW
                                      LD      (P3),HL
                                      IFDEF   ADW
                                      IF      ADW LT $
                                      EXITM
                                      ENDIF
                                      ENDIF
                                      LOCAL   EM
                                      JR      EM
                              ADW:    ADD     HL,DE
                                      RET
                              EM:
                                      ENDM
                              ;
 0000'                        AA:     DS      2
 0002'                        BB:     DS      2
 0004'                        CC:     DS      2
                              ;
                                      ADDW    AA,BB,CC
 0006'    2A 0000'        +           LD      HL,(AA)
 0009'    ED 5B 0002'     +           LD      DE,(BB)
 000D'    CD 0015'        +           CALL    ADW
 0010'    22 0004'        +           LD      (CC),HL
 0013'    18 02           +           JR      ..0000
 0015'    19              +   ADW:    ADD     HL,DE
 0016'    C9              +           RET
 0017'                    +   ..0000:
                                      ADDW    BB,CC,AA
 0017'    2A 0002'        +           LD      HL,(BB)
 001A'    ED 5B 0004'     +           LD      DE,(CC)
 001E'    CD 0015'        +           CALL    ADW
 0021'    22 0000'        +           LD      (AA),HL
                                      ADDW    CC,AA,BB
 0024'    2A 0004'        +           LD      HL,(CC)
 0027'    ED 5B 0000'     +           LD      DE,(AA)
 002B'    CD 0015'        +           CALL    ADW
 002E'    22 0002'        +           LD      (BB),HL
                              ;
                                      END
 Macros:
 ADDW
```

3.9 ソートもできる——マクロの組み合わせ

2.7.3で**IRPC**について説明しました．このとき例としてレジスタ・ペアの**PUSH**や**POP**を1行で表現するマクロを定義しました(図2.20，図2.21)．この時はレジスタ・ペアを**ABDHXY**の各1文字で表現するためのものでしたが，これをさらに進めて，各文字がどのような順序で指定されても**PUSH**される時と**POP**される時に対応が崩れないようにできるでしょうか．もし可能であれば，かなり複雑な構文解析のようなことまでできることになりそうですが……．

方法はいろいろありましたが，結果的には図3.19に示したものが最も少ないマクロ定義になりました．**PUSH**と**POP**の合計で19行，片方では12行で図2.21のものよりも少なくなっています．マクロは二重になっていて，実際に**PUSH**または**POP**を作り出す部分は**TESTP**という名前で，4つのパラメータが与えられます．各仮パラメータの関係は，**P**と**R**が一致していれば**Q**を**U**するというものです．**Q**はレジスタ・ペアの正式名称，**U**は**PUSH**または**POP**というわけです．

この**TESTP**を呼び出す側は，**PUSH**についていえば最初に**IRP**が**IRPC**に対して**A**と**AF**を与えます．**IRPC**は，呼び出し時の実パラメータの文字列の中に**A**があれば**AF**を**PUSH**してくれるマクロ**TESTP**を文字数の回数だけ呼び出します．**IRPC**が実パラメータのスキャンを1回終わるとその外側の**IRP**が，**B**と**BC**を与えて同じことを繰り返します．

PUSHと**POP**の違いは，1文字表現と正式名称との組み合わせが**IRP**で展開される順序を逆にしてあることです．この方法では**A**と**AF**の組み合わせを書いてありますから，この部分を変更すれば同じパターンで全く別の用途にも使えそうです．

展開結果は正しくソートされています．

〔例題㉖〕 図3.19のマクロ定義の呼び出し時に，

PUSHS…HADAH

と書いたらどうなるか．

〔解答〕 **AF，AF，DE，HL，HL**と5つの**PUSH**命令が作られる．

〔例題㉗〕 同じレジスタを2度以上**PUSH，POP**しないためにはどのように変更すればよいか．

〔解答例〕 マクロ**TESTP**の3行目**U Q**の次に**EXITM**を入れる．

〈図 3.19〉 **PUSH** と **POP** を1行で表現するマクロ

```
                .Z80
        PUSHS   MACRO   REG
                IRP     P,<<A,AF>,<B,BC>,<D,DE>,<H,HL>,<X,IX>,<Y,IY>>
                IRPC    R,REG
                TESTP   P,R,PUSH
                ENDM
                ENDM
                ENDM
        TESTP   MACRO   P,Q,R,U
                IFIDN   <P>,<R>
                U       Q
                ENDIF
                ENDM
        POPS    MACRO   REG
                IRP     P,<<Y,IY>,<X,IX>,<H,HL>,<D,DE>,<B,BC>,<A,AF>>
                IRPC    R,REG
                TESTP   P,R,POP
                ENDM
                ENDM
                ENDM

                PUSHS   AHDX
0000'  F5     +         PUSH    AF
0001'  D5     +         PUSH    DE
0002'  E5     +         PUSH    HL
0003'  DD E5  +         PUSH    IX
                        POPS    DXAH
0005'  DD E1  +         POP     IX
0007'  E1     +         POP     HL
0008'  D1     +         POP     DE
0009'  F1     +         POP     AF

                        PUSHS   ADBYXH
000A'  F5     +         PUSH    AF
000B'  C5     +         PUSH    BC
000C'  D5     +         PUSH    DE
000D'  E5     +         PUSH    HL
000E'  DD E5  +         PUSH    IX
0010'  FD E5  +         PUSH    IY
                        POPS    BDAHYX
0012'  FD E1  +         POP     IY
0014'  DD E1  +         POP     IX
0016'  E1     +         POP     HL
0017'  D1     +         POP     DE
0018'  C1     +         POP     BC
0019'  F1     +         POP     AF
                        END
```

3.10　文字列の中の1文字を取り出す――IRPCの応用

メモリ内の1バイトを別の番地へ移すマクロを **MOVM** とします．

```
MOVM    MACRO  A1, A2
        LD     A, (A1)
        LD     (A2), A
        ENDM
```

これで，**MOVM　ADS1，ADS2** としてメモリ内のデータが移せますが，リテラル・データを移すことはできません．**MOVM　0, ADS2** と書けば，0番地の内容が **ADS2** に入ります．意図していることは0というデータを **ADS2** へ書きたいのですが．

それならマクロ定義の **A1** と **A2** についているカッコを取れば **MOVM　0, (ADS2)** として可能になります．しかし，ここでカッコを書きたくないのです．カッコが目ざわり（カッコわるい）なのです．そこで，実パラメータの最初の文字が数字であればカッコなしの **LD　A,** を，数字でなければカッコ付きの **LD　A,** を作るようにできればと思います．そこで，**IRPC** の文字分解機能と，**EXITM** の展開中止機能を使います．

2.10節の宿題で無条件の **EXITM** の効果云々となっていますが，最初の文字の検出がその答です．

```
        IRPC   Q, ABCD
CH      DEFL   '&Q'
        EXITM
        ENDM
```

この **IRPC** は本来4回展開されるべきところが，最初の時に無条件で **EXITM** に出会うので仮パラメータ **Q** に **A** を与えた1回だけで展開を中止します．**CH** の **DEFL** は有効ですから，この例では **CH** は **41H** という値を持っています．すなわち，実パラメータの先頭文字のASCII値が入ってくるのです．

これを利用すれば数字なら値，文字ならシンボルと考えてアドレスの内容をロードすることができます．図3.20がその定義と展開例です．シンボルと数値の判断はアセンブラ自身も先頭の文字で行っており，16進数のように途中にA～Fや最後にHがつく場合でもアセンブラの判断とこのマクロの判断とは一致します．

〈図3.20〉リテラルとアドレスを判断するマクロMOVM

```
                                     .Z80
                            MOVM     MACRO     A1,A2
                                     IRPC      Q,A1
                            CH       DEFL      '&Q'
                                     EXITM
                                     ENDM
                                     IF        CH GT 58 ;Numer ?
                                     LD        A,(A1)
                                     ELSE
                                     LD        A,A1
                                     ENDIF
                                     LD        (A2),A
                                     ENDM
                            ;
0000'                       ABC:     DS        1
0001'                       BCD:     DS        1
                            ;
                                     MOVM      53H,ABC
0002'    3E 53        +              LD        A,53H
0004'    32 0000'     +              LD        (ABC),A
                                     MOVM      ABC,BCD
0007'    3A 0000'     +              LD        A,(ABC)
000A'    32 0001'     +              LD        (BCD),A
                            ;
                                     END
```

別の応用例として，コンソールへ文字列を出力するためのマクロを考えてみます．マクロ名を **MESAG** として，

```
MESAG    メッセージのアドレス
MESAG    'KONNICHIWA'
```

のような2種類の呼び出し方に対応できるようにしたいので，最初の文字が〝'〟であるかどうかを **IRPC** でチェックします．そこで，

```
MESAG     MACRO     DATA
          IRPC      Q, DATA
CH        DEFL      '&Q'
          EXITM
          ENDM
          IF        CH  EQ ' ' '
```

(リテラルの場合の処理)

ELSE

(アドレスの場合の処理)

ENDIF

⋮

〈図3.21〉引用符が対になっていないために起こったエラー

```
                                        .Z80
                                MESAG   MACRO   DATA
                                        LOCAL   MAD,ADS
                                        IRPC    Q,DATA
                                CH      DEFL    '&Q'
                                        EXITM
                                        ENDM
                                        IF      CH EQ ''''
                                        JR      ADS
                                MAD:    DB      DATA,'$'
                                ADS:    LD      DE,MAD
                                        ELSE
                                        LD      DE,DATA
                                        ENDIF
                                        LD      C,9     ;cp/m str out
                                        CALL    5       ;cp/m IO ads
                                        ENDM
                                ;
                                        MESAG   'KONICHIWA!'
O 270D                      +   CH      DEFL    ''''
O                           +           IF      CH EQ ''''
  0000'   18 0B             +           JR      ..0001
  0002'   4B 4F 4E 49       +   ..0000: DB      'KONICHIWA!','$'
  0006'   43 48 49 57       +
  000A'   41 21 24          +
  000D'   11 0002'          +   ..0001: LD      DE,..0000
  0010'   0E 09             +           LD      C,9     ;cp/m str out
  0012'   CD 0005           +           CALL    5       ;cp/m IO ads
                                        MESAG   SAYO
O                           +           IF      CH EQ ''''
  0015'   11 001D'          +           LD      DE,SAYO
  0018'   0E 09             +           LD      C,9     ;cp/m str out
  001A'   CD 0005           +           CALL    5       ;cp/m IO ads
                                ;
  001D'   53 41 59 4F           SAYO:   DB      'SAYONARA'
  0021'   4E 41 52 41
                                        END
```

という形式でマクロを定義して呼び出し例をアセンブルしたところ，アセンブル結果は正しいのですが，エラーが出ています．図 3.21 がそのリストで，O エラーです．これは引用符が対になっていないために起こっています．このマクロははじめに，

IF　　　CH　EQ　' ' ' '

として試していたのが正しく働かないので〝'〞をひとつ減らして同じ形式にしたものです．

このように，実パラメータとしてリテラルの文字列が書かれた時にはエラーになります．前の図 3.20 でも，

MOVM　　'A', ADS2

として呼び出せばやはりエラーになります．こちらの方は **CH** の値が 58 より大きいかどうかを判断していますから，判断結果は大きいとなり，

LD　　A, ('A')

のようになって，意図通りになりません．

〔例題㉘〕 図 3.20 で，**'A'**のようなリテラル文字を数字と同じ処理になるようにしたい．どうすればよいか．

〔解答例〕 **CH　GT　58** を **LOW　CH　GT　58** に変更すればよい．図 3.21 で最初の O エラーの行に **270D** という値が出ている．すなわち〝' ' '〞はこんな値に評価されていることになる．そこで，この下 8 ビットを **LOW** 演算子で取り出せば 58 より小さくなって数字の仲間であると判断される．

〔例題㉙〕 図 3.21 でエラーが発生せず，しかも正しい動作ができるようにするにはどんな方法があるか．

〔解答例〕 **CH　DEFL　'&Q'** を **CH　DEFL　'&Q&Q'** として，**CH　EQ　' ' '** を **CHEQ　' ' ' '** に変更すればよい．一致だけを見るにはこの方法が使えるが，前問のように大小比較になると一考を要す．

〔例題㉚〕 最初の文字でなく，最後の文字を **CH** に与えて出てくるにはどうすればよいか．

〔解答例〕

```
        IRPC    Q, 文字列
CH      DEFL    '&Q'
        ENDM
```

要するに全部展開すればよい．最後に残って出てくるのが最後の文字だから．

〔宿題〕 **LDIR** 命令は強力な命令だが，その場で起動するには各レジスタに値を与えるのに3行必要になる．そこで **LDIR** をマクロにして，

LDIR　　HLの値，DEの値，BCの値

のように表現したい．各パラメータは省略可能とし，省略した時はレジスタのロードそのものをなくする．もしすべて省略した時には，**LDIR** のみ書くことになり，本家 **LDIR** と全く同じ内容になる．このように呼び出せるマクロ **LDIR** を定義せよ．

3.11 逆アセンブル防止用マクロ

プログラム保護に関する法律も検討されていますが，自己防衛対策も無駄ではありません．ソース・プログラムのレベルでは，ハードウェアに依存する部分やノウハウの含まれている部分をマクロに定義し，公表するのはシステム上の動きがわかるだけのマクロ呼び出し部分のみにすれば簡単にコピーできません．リストが必要であればマクロ展開をリストしないようにもできます．マクロ定義部は当然リストしないようにしておきます．

しかし，実際に作られたプログラムがROMやファイルで実行可能になっている場合には，そのROMからコピーが作れます．また，逆アセンブルして解読し，順序を入れ変えたりして一見コピーでないように見せる手もあります．そっくりコピーされるのは仕方ないとしても，逆アセンブルが防止できれば少しは役に立つでしょう．

とはいっても，そのために毎回頭をひねっていたのでは大変ですからマクロ命令を作ってケムに巻こうというわけです．その中身は一般に知られた方法ですが，これがマクロになっていると頻繁に入れることができ，効果が上がります．その分，メモリ・エリアが喰われる点は負担しなければなりません．方法は **JR** や **JP** の無条件分岐命令の後に2～4バイト命令の命令コードだけを入れるものです．

〈図 3.22〉 プログラム保護のためのマクロ **JMR, JMP**

```
                                   .Z80
                          JMR      MACRO    ADS
                                   DB       18H,ADS-$-1,21H
                                   ENDM
                          JMP      MACRO    ADS
                                   DB       0C3H
                                   DW       ADS
                                   DB       3AH
                                   ENDM
                          ;
0000'   3E 05             STT:     LD       A,5
                                   JMR      ABC
0002'   18 01 21      +            DB       18H,ABC-$-1,21H
0005'   01 01F4           ABC:     LD       BC,500
                                   JMP      BCD
0008'   C3            +            DB       0C3H
0009'   000D'         +            DW       BCD
000B'   3A            +            DB       3AH
000C'   09                         ADD      HL,BC
000D'   C3 0000'          BCD:     JP       STT
                          ;
                                   END
```

〈図 3.23〉 図 3.22 を逆アセンブルした例

```
L100,10F
  0100  LD   A,05
  0102  JR   01
  0104  LD   HL,F401
  0107  LD   BC,0DC3
  010A  NOP
  010B  LD   A,(C309)
  010E  NOP
  010F  NOP
  0110
```

マクロ名に **JR** と **JP** をそのまま使いたいところですが，そうするとすべての **JR**，**JP** が 1 バイト余分にメモリを喰うので，プログラマがコントロールできるように **JMR** と **JMP** にしました．そのマクロ定義と展開例が図 3.22 です．また，これを逆アセンブルしたのが図 3.23 に示してあります．この場合は，かなりよくできた例で，逆アセンブルからは元のソース・コードの見当がつきません．しかし，よく見ると 104 番地の **LD HL…**の命令コードのみが **JR 01** で飛び越されていることがわかります．

筆者もプログラムの解析に逆アセンブルやサーチを使用しますが，もしこのような防止策がなされていれば解析の手間は数10倍かかります．そして，おそらく諦めるでしょう．しかし，世の中には時間の余っている人も少なからずいたりしまして，**JP**の後に**21H**が，**JP**の次には**3AH**が入っているらしいと判断すれば，コンピュータを使って取り除く（通常**NOP**にすれば悪さをしなくなる）でしょう．

そこで，さらに高度なキツネとしては，追加する命令コードをその都度変更するようにします．バイト数も変えられます．さらに，無条件分岐のところに条件分岐を使って，まさかと思われるところまでダマシ命令を入れたりします．こうなると，さすがのタヌキもお手上げになります．このように対策されたバイナリ・コードを解析する手数は，無対策のときの数百から数千倍になると思われます．プログラムの大きさは1.2~1.5倍くらいには入ります．

〔例題㉛〕 **JR**と**JP**をマクロ名にした場合，**JR**と**JP**がすべてダマシ命令を伴う点以外の問題点は何か．

〔解答例〕 **JR**や**JP**には条件分岐のものがあり，命令コードだけでは区別がつかない．そこでオペランドにパラメータが2つあれば条件付きであるのはわかるが，その条件によって**0C2H**や**0CAH**など個々に命令コードを書かなければならない．かなり大きなマクロになる．

第４章
リロケータブル・マクロ・アセンブラによるソフト開発

前章までは，主としてマクロ定義と呼び出しに関する手法，いわゆるコーディング・テクニックについて述べましたが，ソフトウェアの開発はコーディングだけでは成立しません．大きなソフトウェアでは，作業分担や進行管理という部分まで含めてさまざまな要素がからんできます．

ソフトウェア資源の保存管理についても，冷蔵か，冷凍か，塩漬けか……というわけでもありませんが，保存方法がいくつか考えられます．というのは，CP/M80に組み込まれているASMのようなアブソリュート・アセンブラでは，単一のソース・ファイルのみを入力としていましたから，バイナリ・コードのレベルで管理するか，ソース・プログラムで管理するかの2種類しかなかったのですが，リロケータブル・マクロ・アセンブラではオブジェクト・コードのレベルで結合したり選択分離したりできるようになって，管理方法も多くの要求に対応できるようになります．

しかし，一方では操作が複雑になり，アセンブル作業の時間もかかりすぎるということもあります．たとえば，ASMでアセンブルすると，即，**.HEX**ファイルができあがり，そのままROMライタにもDDT（デバッガ）にも入力できるのに対し，M80やRMACなどではリンクという手続きを経なければ箸にも棒にも掛からないのです．第一，1本のプログラムでもリンク（鎖のようにつなぐこと）するという意味がよくわかりません．

というわけで，せっかくのリロケータブルなシステムもある程度の利用上の知識がなければその機能を発揮させることができません．実際，マクロ機能のないアセンブラでは，リロケータブルな複数モジュールを作り出して，アセンブル後にリンクするという手続きが面倒で，ソース・プログラム上で結合してしまいがちです．ところがマクロ機能を利用すると，このような手続きについてもその都度意識するわずらわしさがほとんど解消します．リロケート機能＋マクロ機能でソフト開発の最強のシステムができ上がるのです．

本章ではM80システムの特長とその利用法について，マクロ機能との関連性および，より有効な活用法をさぐります．

4.1 リロケータブルの有用性

M80に比べて簡単なアセンブラをCP/M80に付属のASMとすると，後者によるソフトウェアの開発システムは図4.1のようになります．ASMのように絶対番地でオブジェクト・コードを出力するものをアブソリュート・アセンブラと呼びますが，その名の通り，アセンブルされたコードは番地を移動することができません．また，アセンブラに入力できるファイルは単一のソース・ファイルのみですから，アセンブル段階で別のファイルのサブルーチンを結合することもできません．

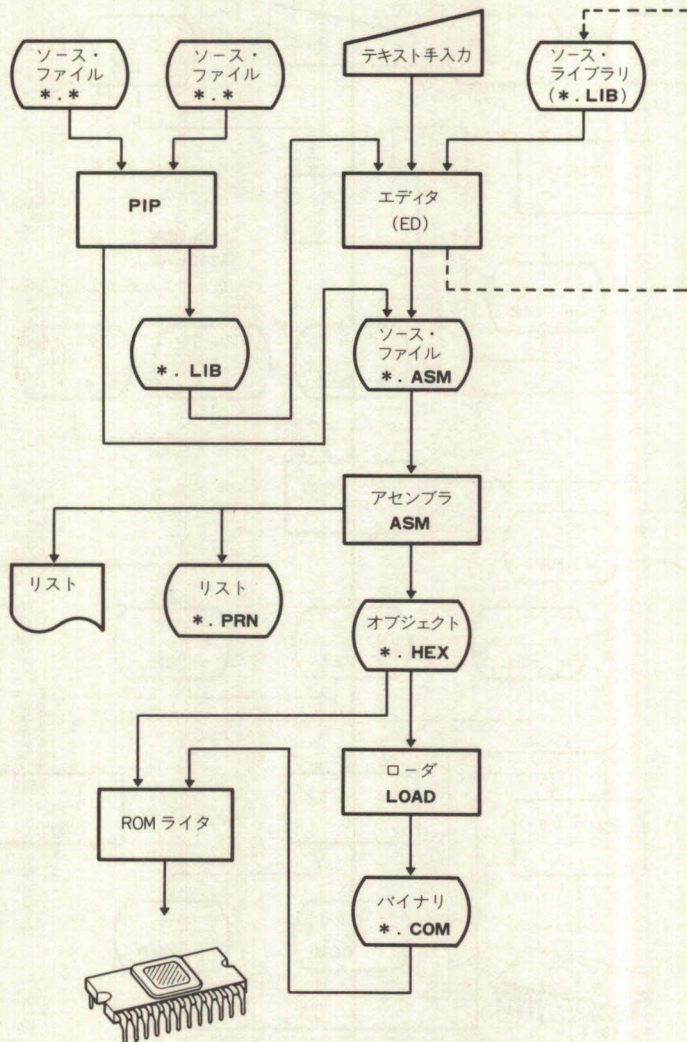
サブルーチンの結合は，基本的にはすべてソース・ファイル作成段階で行わなければならず，エディタで扱うファイルが非常に大きなものになってしまう可能性があります．すでに完成しているサブルーチンを，エディタ操作上のミスで正しく使えなくしてしまうことも起こるでしょう．

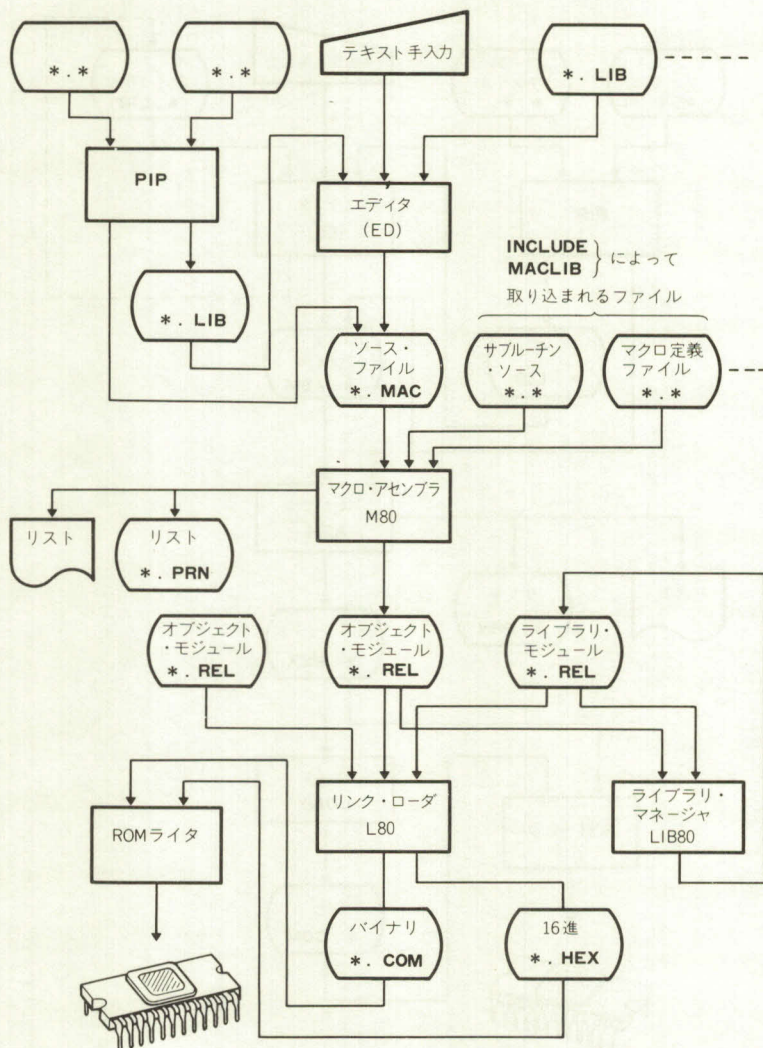
リロケータブル・マクロ・アセンブラによる場合は図4.2のようなシステムになり，ファイルの結合が3つのレベルでできます．すなわち，まずASMと同じエディタのレベル，次にアセンブル時に自動結合されるソース・ファイル，そしてアセンブルの結果出力されるリロケータブル・オブジェクトのレベルです．

アセンブル時に自動結合されるのは，メインとされるソース・プログラムの中にソース・ファイル結合指示(**INCLUDE**など)が書けるからです．この機能はマクロともリロケータブルとも直接関係はありませんが，マクロの中からそのマクロが呼び出された時，またはさらにそのマクロに特定の実パラメータが与えられた時に必要なファイルを結合することができます．これによって，余分なファイルを結合したり，必要なファイルをつなぎ忘れたりといった間違いは大幅に減ります．

マクロ機能によって，簡単な機能も大きなメリットに生まれ変わってくるわけですが，それにしてもソース・ファイルのレベルで結合したのではミスは減ってもアセンブル時間は減りません．やはりリロケータブルの機能を最大限に発揮するのはアセンブル後に結合できることです．

アセンブル後に結合するからといって，ソース・プログラム上にあらかじめジャンプ先のリストを書いておくとか，メモリが詰まって動きが取れなくならないようにスペースを空けておくといった必要は全くありません．リロケータブルのオブジェクト・コードをリンカによって結合する場合でも，ソース・レベルでつないだ場合と比べてメモリの使用量が1バイトも増えることはありません．むしろ，アドレスの配置をリンカが解決してくれ

〈図 4.1〉 アブソリュート・アセンブラによるソフトウェア開発のシステム

るので無駄がなくなるくらいです.

〈図4.2〉リロケータブル・マクロ・アセンブラによるソフト開発システム

4.1.1 グローバル・リファレンス

別のオブジェクト・コードが結合できるというのはどういうことでしょうか．当然，データやプログラムの番地がお互いにわかるようになっていなければならないのに，リロケータブルということで，オブジェクトの段階では番地が決まっていないのです．決まっていない番地をお互いにわかるためにはアセンブルし直すと同じ手間がかかる……？

全くその通りですが，リロケータブルなコードといえども，各モジュールごとに先頭から何番地離れているかを表す数値がすでに決まっているのです．ということは，各モジュールの大きさを加算すれば，各々の先頭が何番地になるか計算できます．これによって，個々のラベルなどの値を計算してオブジェクト段階では不明であった参照番地を適切に埋めて行く……これがリンクするといわれる作業内容です．

この時，オブジェクト段階で不明になっているシンボルをグローバル・リファレンスと呼びます．結合するべきファイルを忘れると，グローバル上の未定義状態が発生します．これはリンカがエラー表示をします．それにしても，リンカでこれだけの作業ができるためには，アセンブラがオブジェクト・コードの中に多くの情報を書き込んでおかなければなりません．

アセンブラは，プログラム内で決定されたシンボルの値をすべてオブジェクト上に出力するのでしょうか．また，そのモジュール(モジュールは一度にアセンブルするプログラムの単位,大きさにかかわらない)の中で決定されていないシンボルをすべてグローバルの中から捜させるのでしょうか．

そうではありません．各モジュールの中で，グローバルから捜し出すべきシンボルと，グローバルとなって他のモジュールに値を与えるべきシンボルを明確に宣言しなければならないのです．これは一見面倒なようですが，これには深い訳があります．

第一に，そのモジュール内で定義されていないシンボルを，即外部から捜さない理由は，内部に書くべきものを書き忘れたか，書き間違えている可能性が考えられるからです．また，間違えてもグローバルになければエラーになりますが，たまたまグローバルで見つかってエラーにならなければそのプログラムはデバッグに入ってからはじめてミスが見つかることになります．エラーは早く発見されるに越したことはありません．

第二に，どのモジュールからでも見えるグローバルを増やすと，グローバル同士の重複が起きやすいことです．オブジェクト・モジュールの大きさや，リンカの処理時間の都合などで，グローバル・リファレンスの文字数はL80では6文字までに限定されています．アセンブル段階で6文字以後は切り捨てられます．しかも，グローバルですから完成され

たプログラム全体の中で重複が許されません．重複を避けるために，グローバルに出すシンボルは必要最小限に止めなければなりません．

第三に，簡単なことですが，オブジェクト・モジュールを単に小さくするためという理由もあります．

4.1.2　グローバル宣言

M80では，外部から見えるようにするための宣言と，外部から捜し出して決定されるための宣言と別れています．外部から見えるようにするためには，

```
GLOBAL  JP1, P8255, CTC
```

のように書きます．**JP1**や**P8255**は内部で定義されていなければ，未定義のエラーになります．また，**GLOBAL**は**ENTRY**および**PUBLIC**とも書けます．三者は全く同等です．外部に値を求めるものは，

```
EXTERNAL    PIO, SHORI
```

と書きます．**PIO**や**SHORI**は内部で定義されていると
エラーになります．**EXTERNAL**は，**EXT**および**EXTRN**とも書け，全く同じ扱いになります．

各々3種類の宣言疑似命令が用意されているのは，他のアセンブラとの互換性をよくするためのもので，その他には特に意味はありません．M80ではこの他に，疑似命令によらない宣言法があります．これは実際に便利な機能です．

```
JP1::    CALL    SHORI##
```

と書くと，**JP1**が外部から見えるように，**SHORI**を外部から捜し出すように宣言されます．コロン：ふたつが**GLOBAL**の意味，メッシュ＃ふたつが**EXTERNAL**の意味を持っているのです．ただし，**EQU**されるシンボルは疑似命令によってしか外部から見えるようにできません．

4.1.3　シンボルの重複防止

プログラムをモジュールに分割してアセンブルするメリットは，エディタやアセンブルの時間を節約するだけでなく，シンボルの多重定義の防止に役立ちます．各モジュール内でグローバルになっていないシンボルは，他のモジュール内のシンボルと一致していても

〈図 4.3〉 1 バイトずつ PC を変更したプログラム

```
0000'                   ASEG
                        ORG     100H
0100    03              DB      3
0101                    CSEG
0000'   05              DB      5
0001'                   DSEG
0000"   07              DB      7
                        COMMON  /AB/
0000!   09              DB      9
                        END
```

個別にアセンブルされるのでエラーにはなりません．しかも内部でのみ使用されるこれらのシンボルは，16 文字まで識別されるのでかなり大胆な表現ができます．

とはいっても，モジュールを小さく区切りすぎると，モジュール間での参照が増えるようになり，グローバルなシンボルが増えてしまいます．これでは何も意味がなくなりますから，モジュールはある程度まとまった処理の単位でなければなりません．

4.1.4 プログラムの ROM 化

アブソリュート・アセンブラでは，ROM のプログラム・エリアについては順に積み重ねて行けますが，RAM の割り付けに関しては，あらかじめ全体で使用する番地を考えておかなければなりません．リロケータブルであっても，単一の PC しか持っていなければ同じことがいえます．両方とも，あらかじめ RAM のエリアを決めなければ，プログラムの中にデータ用の RAM であるべき番地が混じっており，分離することができません．

M80 では簡単に ROM と RAM に分離できるように，4 つの PC（プログラム・カウンタと呼ぶのが一般的だが，データ・エリアのことを考えると，ポジション・カウンタの方が合っている）を持っていて，ひとつはアブソリュートのカウンタ，残る 3 つはコード，データ，コモンのリロケータブルなカウンタになっています．PC は，シンボル：　または　シンボル：：　と書いた時にシンボルに与えるべき値をカウントしているものですが，これが独立に 4 本用意されているので，リンクする時にデータ同士，コード同士，コモン同士をすべてまとめることができます．アブソリュートはまとめることはできません．その名のとおり，アセンブル時点で絶対的に固定された番地として定義されているからです．

4 つの PC によって同じ数値でも 4 種の意味を持つわけですが，シンボルの値を与えた PC をそのシンボルのモードと呼びます．すなわち，各シンボルは内部でも，グローバルで

〈図4.4〉図4.3の無指定リンク

```
L80 B:F6,B:F6,B:F6/N/E

Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft
%Overlaying Data area

Data     0100     0108     <     8>

43044 Bytes Free
[0000    0108          1]

DDT F6.COM
DDT VERS 2.2
NEXT  PC
0180 0100
-D100,107
0100 03 00 00 09 07 05 07 05 ........
```

も数値の他にモードの区別を持っているのです．アセンブルによって生成されたコードもそのときのPCのセグメントに割り当てられます．

たとえば，図4.3のように，1バイトずつPCを変更したプログラムでは，アブソリュート(**ASEG**)の**100H**以外は，コード(**CSEG**)，データ(**DSEG**)，コモン(**COMMON**)とも0番地です．このオブジェクト(全く同じ内容のもの)を2度リンクして，**.COM**ファイルを作り，その内容をDDTで見たのが図4.4です．リンク時のオプションは，**.COM**ファイルを作るためだけのもので，その他のリロケーションのためのオプションは指定していません．

これを見ると，アブソリュートは絶対番地だから同じ**100H**に重なってしまい，警告が出ています．また，コモンは同じ名前なのでその名のとおり共通エリヤとなってやはり重なります(これはエラーではなく，そうなるべくしてなっている)．その後，**07**と**05**が2度書かれています．コモンが**103H**から始まるのは，リンカが無指定では**103H**からと自動的に決めることによります．

ところで，アブソリュート・セグメントとコード・セグメントをROMに，コモンとデータ・セグメントをRAMに割り付けたい時，このままでは困ります．そこで，図4.5のようにオプションを追加して，最終的な(アセンブル時には決まっていない)番地を指定します．すると，ROM部分とRAM部分が分かれた形にリンクされます．もちろん，オブジェクト・コードは図4.4と全く同じで同一のファイルを2度結合したものです．

このようにして，ROM，RAMをどのアドレスにも指定でき，それによって割り付け番

〈図 4.5〉 図 4.3 の番地指定リンク

```
L80 /P:101,/D:104,F6,F6,F6/N/E

Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft
%Overlaying Program area

Data      0104    0107    <    3>
Program  0100    0103    <    3>

43053 Bytes Free
[0000    0107          1]

DDT F6.COM
DDT VERS 2.2
NEXT  PC
0180 0100
-D100,106
0100 03 05 05 00 09 07 07 .......
```

〈図 4.6〉 リンカ L80 による ROM 化可能な再配置

モジュール 1　モジュール 2　リンカによって作られるメモリ上の配置

ASEG CSEG DSEG COMMON ASEG CSEG DSEG COMMON

0000　FFFF　アブソリュート　リロケータブル　リロケータブル　アブソリュート

/P: nnnn　/D: vvvv　Ⓓ, Ⓗ

アブソリュート 2　アブソリュート 1　コード 1　コード 2　ROM に書く

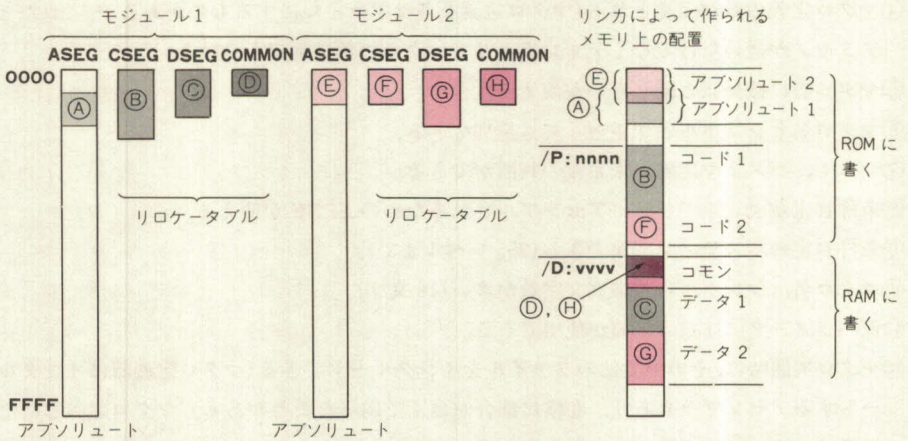
コモン　データ 1　データ 2　RAM に書く

地が変更されるだけでなく，この例ではわかりませんが，すべての参照番地が正しく決定されます．アブソリュート・セグメントだけはリンカによって変わりませんから，ユーザの責任で管理しなければなりません．しかるべき管理をすれば，ROM 部分としても RAM 部分としても割り付けできます．

図 4.6 はリンカの作用を概念的に表したものです．通常はこのように各セグメントが重

ならないように **nnnn** と **vvvv** を指定しますが，重なってもリンカは警告をするだけでリンク作業は中止しません．重なる部分についてはリンカは保証しませんが，それ以外は保証されます．

従来のマクロ・アセンブラよりM80が優れている点

マクロ・アセンブラは古くから使用されていましたが，マクロ機能やその他の機能にも制限が多く，使用範囲はごく限られたものでした．M80はマクロ・アセンブラの中でも非常に充実した機能を持っており，本書におけるマクロ利用法も他のマクロ・アセンブラでは実行不可能な部分があります．M80の優れている点は，

①マクロ定義のネスティングができる．

②マクロ定義の中で，そのマクロ自身を定義できる（自マクロ再定義）．

③マクロ定義の中からのマクロ呼び出し，ネスティング深さが大きく取れる．

④マクロ定義の数が多くとれる(マクロ定義を各々ファイルにするものがあるが，これだとアクセスが遅いだけでなくディレクトリでマクロの数に制限を受ける)

⑤マクロ名に機械語と同じものが使える．

⑥マクロ名とシンボル名が重複しても識別できる．

⑦パス1，パス2や定義，未定義の判断ができる．

⑧条件判定が成立しない時にアセンブルされるための **ELSE** が使える．

⑨条件判定のネスティングができる(255レベルまで)．

⑩マクロ名，シンボル名の識別文字数が多い(16文字)．

⑪仮パラメータに任意の名前が使用できる．

⑫マクロ展開時に，その中で他のファイルをインクルードできる(マクロ定義時にインクルードするアセンブラもあり，非常に都合が悪くて困ったことがある．マクロ定義しただけでソース・プログラムが大きくなってしまう)．

⑬マクロ再定義で古い定義は消される．再定義は何回でもできる．

⑭マクロ展開時に再現されるコメントと，再現されないコメントが作れる．

⑮PCを4個持っている．

⑯オブジェクト・コードは，多くの他のコンパイラのオブジェクトと互換性があり，リンクできる．

などです．

M80への要望

M80は優れたアセンブラですが，欲をいえば，

①コメントと引用符中だけでもカナ文字を使えるように

```
DB    'リテラル'  ;コメント
```

②大文字と小文字を区別するように(命令コードはすべて小文字なら大文字と同じと判断してもよい)

③文字列を演算して，結果として文字列を返す関数を使えるように

④ **INCLUDE** のネスティングができるように

コラム

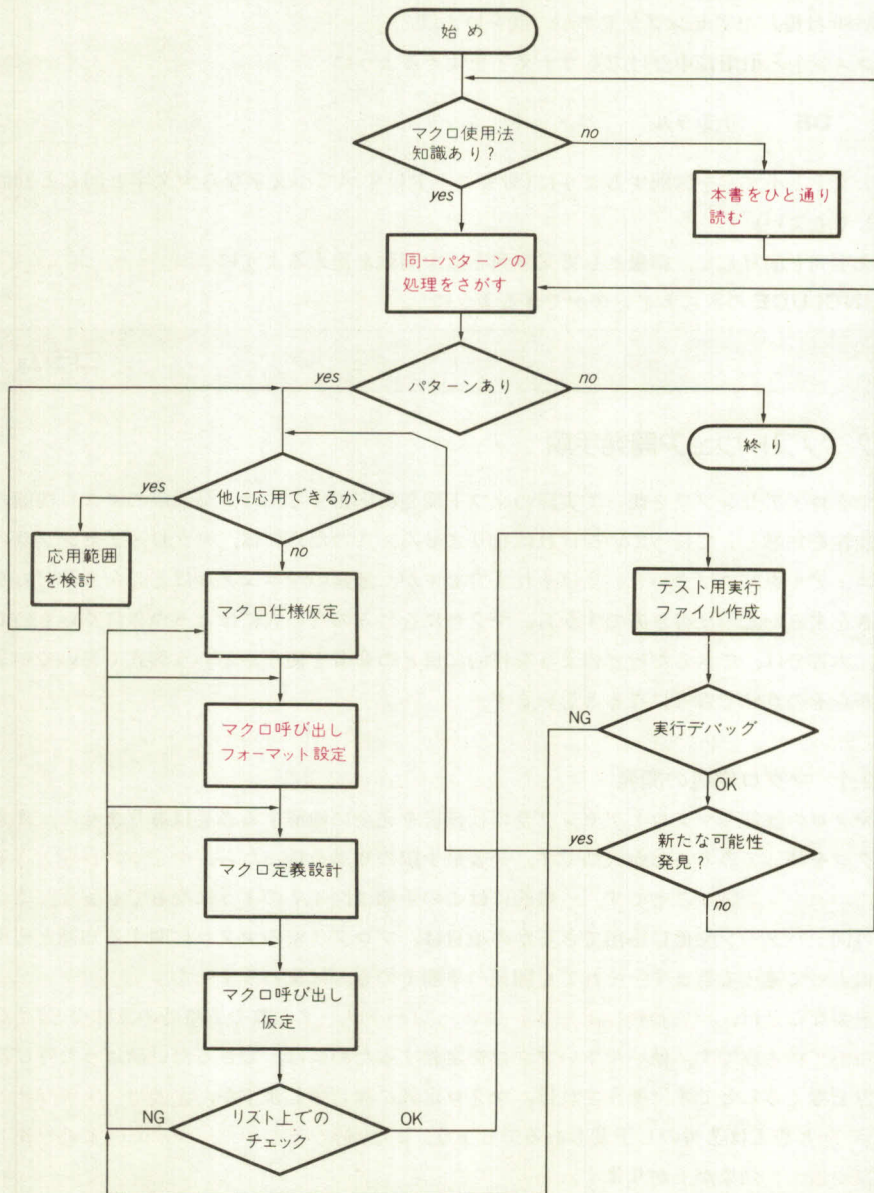
4.2　ソフトウェア開発手順

マクロ・アセンブラを使って実際のソフト開発に応用するには，最低限のマクロ機能の可能性を知識として持っていなければなりません．このためには，マクロ・アセンブラのマニュアルを読めばよいといえばそれまでですが，通常のマニュアルはどの命令がどんな働きをするかという書き方ですから，マクロになじみのない人にはとっつきにくいものです．本書では，できるだけどのような目的にはどの命令を使うかという形式で書いていますからその意味で参考になると思います．

4.2.1　マクロ定義の開発

マクロの機能やマクロ・アセンブラの仕様書を完全に理解する必要はありません．まず，マクロを使ってみることが大切です．必要最少限の知識を持ったら，すぐマクロ化できそうなパターンを捜すことです．一般的にはこの手順は図4.7のようになるでしょう．ここでの同一パターンや他に応用できるかの項目は，プログラマのマクロに関する知識と応用力によって違ってきます．それでも開発の手順そのものは変わりません．

重要なことは，マクロ呼び出しフォーマット(パラメータの数や省略時の意味など)を先に決めている点です．使いやすいマクロを定義するためには，できるだけ欲ばった呼び出し方を考えることです．そうすれば，マクロ定義には苦労しますが，定義は一回呼び出しは N 回と思えば苦労のし甲斐もあるでしょう．また，呼び出しやすいマクロほど呼び出し回数が増えて効果が上がります．

〈図4.7〉 マクロ定義開発手順

マクロを使っている場合には，アセンブラがエラーを検出しなくてもパラメータの伝達が正しく行われているとは限りません．また，場合によってはエラー表示のままでも使えるマクロ(図 3.21 参照)もあります．したがって，オブジェクトを出す前にリスト上で確認する必要があります．条件判断や省略時解釈を使用しているときは，一応すべての組み合わせに対して展開例が得られるようにマクロ呼び出しのパラメータを変えてみることも大切です．実行デバッグに入ってから，マクロ展開がパラメータによっておかしくなったりすると，見つけ出すのは至難です．

リスト上での確認が終わったら，マクロ部分のみの実行デバッグを行えばより完全ですが，いきなり全体の実行に入ってもかまいません．実行デバッグに入ってから細部の修正や変更をやっていると，プログラムの流れが似ているような箇所が新たに発見されたりします．このようにして気付いたところをメモしておき，新たにマクロにできないか，他のマクロに含めて応用範囲を拡げられないかを検討してみると最初の段階では気付かなかった要素が多いものです．

4.2.2 マクロ・ファイルの保存管理

マクロ定義が完全なものになったら，マクロ定義部分だけを別のファイルにしておいて，メイン・プログラムから **INCLUDE** によってアセンブル時に結合すればよいでしょう．または，最初からマクロ定義を別ファイルにして作成する方法もとれます．ただし，別ファイルになっている場合は，デバッグ中にエラーを見つけて変更するファイルを間違えることが多いので注意が必要です．慣れないうちは，１本のファイルになっていた方が楽かも知れません．

デバッグ済みのマクロ定義が増えてくると，マクロ定義ファイルが大きくなって取り扱いにくくなります．M80 では，マクロ定義された内容はすべてアセンブル作業用のメモリに貯えられますが，これがシンボルの対応表と共通になっており，マクロ定義が増えてくると，それだけシンボルの登録できる数が減ります．

マクロ定義ファイルを小さくするには，マクロ定義の種類を分けていくつかのファイルに分割する方法があります．適切な分類ができる場合はこれも一法ですが，分割の仕方が適切でないと結合し忘れてエラーになる率が高くなります．

そこで，マクロ定義の中で一度しか展開されない部分を含んでいるもの(図 3.17，図 3.18 参照）を捜し出して，各々のその部分を別のファイルにします．一度だけしか展開されないのは，多くの場合データ・エリヤか定数，そしてマクロ展開の中から呼び出されるサ

ブルーチンです．別のファイルにしたものは，そのままでは元のファイル(マクロ定義のファイル)から参照できませんから，マクロ定義側と取り出されたサブルーチン側で，必要に応じて **GLOBAL** や **EXTERNAL** の宣言をします．

この時，メインのプログラムで，マクロ呼び出しを使わずに直接サブルーチンを参照したり，データを取り出したりしていなければ，メインのプログラムを変更する必要は全くありません．メインの方では，インクルードするマクロ・ファイルの中にサブルーチンがすべて書いてあるのか，それともサブルーチン部分が別ファイルになっていて別にアセンブルされるのかを知る必要がないのです．結果としてマクロ・ファイルが軽くなっていればアセンブル時間が短くて済むという違いが出てきます．

サブルーチン部分は別にアセンブルされ，別のオブジェクト・モジュールとして保存され，メイン・プログラムのオブジェクト・モジュールとリンクされて実行可能なバイナリ・コードのファイルになります．サブルーチン部をオブジェクト・モジュールで保存すれば，メイン・プログラムのアセンブルが速くなるだけでなく，サブルーチンのファイルが小さくなります．これは，オブジェクト・モジュールがソース・ファイルの約10～20分の1になるからです．

このようにして，マクロ定義部のみをソース・ファイルにし，サブルーチンをすべてオブジェクト・モジュールで保存すれば同一のディスクに多くのマクロが扱えるようになり，開発効率が大幅に上がります．以上の開発ステップを図にすると図4.8のようになります．

この図の(4)のステップではオブジェクト・モジュールを何本か集めてライブラリ・ファイルにもできるとなっています．通常のオブジェクト・ファイルは，1ファイルに1モジュールと限られていますが，ライブラリ作成用プログラムLIB 80を使用すると1ファイルに何個かのモジュールを集めて保存できます．このようにして作られたオブジェクト・ライブラリはリンク時にモジュールごとに必要なものを選び出すオプション **/S** が適用でき，不必要なモジュールでプログラムが大きくなってしまうのを避けられます．

LIB80はこの他に，ライブラリ・ファイルの中から必要なモジュールを取り出して，またさらに別のファイルからのモジュールと結合して新たなライブラリ・ファイルを作成するなどの機能を持っています．オブジェクト・モジュールは内部のコードが一種の暗号のようになっていて，エディタはもちろん，DDTやSIDでも全く解析できません．ライブラリ管理にはLIB80は不可欠です．

(LIB80とオブジェクト・コードのフォーマットについてはユーティリティ・ソフトウェア・マニュアルを参照してください)

〈図 4.8〉 リロケータブル・マクロ・アセンブラによるソフト開発のステップ

(1)

マクロ定義とマクロ呼び出しを含んだソース・プログラム

マクロ使用プログラム作成．マクロ定義と呼び出し，パラメータ渡し受けのチェック（アセンブル・リスト上でチェックする）

(2)

マクロ定義を別のファイルにする

マクロ呼び出しを含むメイン・プログラム．マクロ定義ファイルをインクルードする

マクロ中で使用するサブルーチンのチェック

(3)

毎回展開される部分のみのマクロ定義

マクロ処理用サブルーチン

マクロ呼び出しを含むメイン・プログラム．マクロ定義ファイルをインクルードする

マクロ処理用サブルーチンを別にアセンブルし，リンク段階で結合する．多様なマクロ呼び出しに耐える事を確認

確認ずみマクロ定義をまとめる

使用頻度の低いものは別のファイルに

処理ルーチンはオブジェクト・モジュールで保存．何本かまとめてライブラリ・ファイルにもできる

(4)

毎回展開される部分のみのマクロ定義を集めて共用マクロ定義ファイルとして保存する

上と同じ．特に必要なときのみ使用するオプション・マクロ集

マクロ処理用サブルーチン群
同　上
同　上

完成されたマクロの保存状態

4.2.3　ソフトウェアのデバッグ作業

ROM化されるプログラムも，デバッグの段階ではできる限りOSの下で作業を進めた方が効率がよいことは当然です．M80とL80を使用して作られる**.COM**ファイルはL80に**/P**と**/D**の指定をしなければそのままトランジェント・プログラムとして実行できます．ただし，アブソリュート・セグメントだけは重複しないようにしなければなりません．インタラプトと**RST**命令を使用しないプログラムならこの状態でデバッグできます．ZSIDを使用して58K　CP/Mで約48Kバイトの**TPA**が使えますから普通のプログラムなら実行できます．

入出力命令がそのまま実行できないことが多いので，あらかじめシステムに合わせて仮のI/Oにしておけばターゲット(最終的にROMが搭載される)システムそのままでなくてもテストはできます．入力としてはコンソールのキー入力やシリアル入力を，出力としてはCRTやプリンタなどを使います．それでも足りなければパラレルI/O(8255など)をデバッグ用に追加しておけばだいたい間に合います．

仮のI/Oは，I/O命令がマクロになっていればマクロ定義の変更だけで済み，ROMに書き込む前に，正規のI/Oに変更してアセンブルし直します．

RSTや**NMI**を使用するプログラムでは，CP/MのTPA外のエリアになってしまうので**.COM**ファイルにはできません．この場合は**.HEX**のファイルに作ります．リンカは警告メッセージを出し，**Move Any Way (Y or N)?**と聞いてきますが，**Y**で返答します．これでメモリの中では実際のロード番地とは違う番地でリンク作業が行われますが，出力される**.HEX**ファイルは正しい番地を持っています．

これをDDTで持ってこようとすると，エラー表示をしてロードできません．ところがSID/ZSIDではエラーにはなりますが，ロードできます．これで**RST**については何も問題なくデバッグできます．**NMI**はデバッグ用のシステムのハード構成によって実際に**NMI**信号を入れてデバッグできるとは限りませんが，**NMI**信号を受けてから後の処理だけはデバッグできます．CP/M80のシステムでは8080との互換性の関係から，システムで**NMI**を使用していることはまずありません．

インタラプトのモード2を使用している時は，インタラプト・ベクタのアドレスをTPA内に指定するか，TPA外では0ページ(**0～0FFH**)内に指定しなければなりません．0ページにした場合は**NMI**や**RST**と同じくリンカ，デバッガともエラー表示となりますが実際上は問題ありません．

4.3　M80でカナ文字を扱う方法

M80は非常に優れたアセンブラです．しかし，国産でないためにカナ文字が通りません．内部でビット7を使用しているので簡単に **AND　7FH** を消せばよいというわけには行きません．理想的にはカナ文字でシンボルやマクロ名が書ければいうことはありませんが，それは無理としてもせめてコンソールに出力するメッセージとコメントだけはカナ文字が使えるようにしたいものです．

そこで，M80の中を改造してカナ・コメントだけは通せるのですが，マクロを使用するとだめなようです．いろいろと試みた結果，M80の改造は諦めて，他の方法でカナ文字を通せるように考えてみました．

まず，コメントについては，リスト・ファイル(**.PRN** のファイル)を一度作っておいて，そのファイルを読み取って元のカナ文字に復元する方法が考えられます．リスト上ではソース･ファイル上のカナ文字はビット7を0にした文字に化けています(化かされている)．そこで，コメントの記号(；)が来たらその後の文字のビット7を1に変更し，CRコード(13)が来たらその作業を止めるということができれば，カナ・コメントは再生されます．

これで一見よさそうですが，これだけではカナ以外の文字がコメントに使えなくなるのです．アルファベットも数字もすべてカナに化けてしまいます．これでは困ります．結局両方のコードを通すにはカナ・シフトのコードを作らなければならないというわけで，筆者の場合は ¦ (**7CH**)をシフト・コードに使用しました．¦ を選んだのは，実際に ¦ を書きたいケースは少ないと考えたからです．見えないコードはエディタでの取り扱いが困るのでこれでよしとしました．

コメントは簡単でした．メッセージは大変です．メッセージは，ソース・プログラム上で引用符で囲んで書きます．これをリスト上でカナ文字を再現できるようにするだけなら一応コメントの場合と同じようなもの(実際は同じようではなかった……後述)ですが，実行時に出力される文字は相変わらず化けたままです．そこで，オブジェクト・プログラムから文字列を捜して変更しようと思ったのですが，オブジェクト・コードは全くの暗号で取り扱いが大変すぎます．そこで最終案としては実行時にメッセージ出力ルーチンでビット7を立てる作業を行うことにして落着しました．

幸いなことに，M80では2種の引用符〝'〞と〝"〞が使えます．そこで一般に用いられている〝'〞の方をアルファベット用に，〝"〞の方をカナ文字用に使うことにしました．

もちろん，メッセージ出力はマクロ命令ですから，

MESAG　'COMMENT'
MESAG　"コメント"

のように書けるようにします．"'"か"""かは**IRPC**の最初の文字検出機能(3.10節)で判断します．さらに，カナとアルファベットを混用できるように**7CH**をシフト・キーとして使用します．

ものはついでと，タイトルもカナ文字で表現できるようにしました．タイトルは必ずフォーム・フィード(**FF…12**)コードの後に来ますから，ここで**7CH**が入ればカナに変換します．このような仕様で変換して作られたリストのサンプルが図4.9です．(a)がソース・プログラムで，(b)がM80の出力**.PRN**ファイル，そして(c)がカナ文字復元プリントです．

図4.9は完成後のサンプルですが，実はこのリストの左端に出てくる**0000'**が曲者でした．これは**CSEG**を示す記号なのです．**DSEG**があると"""が出てきます（図4.

〈図4.9〉カナ文字を使用した例

(a) ソース・プログラム

```
;ｶﾅﾃｽﾄ ｶﾅｺﾒﾝﾄ ｵﾖﾋﾞ ｶﾅ|LITERAL|ﾉ ﾃｽﾄ
;|this is A comment
        TITLE   |ｶﾅﾀｲﾄﾙ|
        DB      'DATA'
        DB      '|ｶﾅ ﾘﾃﾗﾙ'
        DB      "ｶﾅ|LITERAL"
        DB      "|LI|ﾃ|RA|ﾙ"
        END
```

(b) M80の **.PRN** ファイル

```
|6E@2DY|            MACRO-80 3.44   09-Dec-81       PAGE    1

                                    ;6EC=D 6E:RJD 5VK^ 6E|LITERAL|I C=D
                                    ;|this is A comment
                                            TITLE   |6E@2DY|
 0000'   44 41 54 41                        DB      'DATA'
 0004'   7C 36 45 20                        DB      '|6E XCWY'
 0008'   58 43 57 59
 000C'   36 45 7C 4C                        DB      "6E|LITERAL"
 0010'   49 54 45 52
 0014'   41 4C
 0016'   7C 4C 49 7C                        DB      "|LI|C|RA|Y"
 001A'   43 7C 52 41
 001E'   7C 59
                                            END
```

(c) カナ文字復元プリント

```
ｶﾅﾀｲﾄﾙ   MACRO-80 3.44    09-Dec-81         PAGE     1


                                     ;ｶﾅﾃｽﾄ ｶﾅｺﾒﾝﾄ ｵﾖﾋﾞ ｶﾅLITERALﾉ ﾃｽﾄ
                                     ;this is A comment
                                             TITLE    |6E@2DY|
  0000'    44 41 54 41                      DB       'DATA'
  0004'    7C 36 45 20                      DB       'ｶﾅ ﾘﾃﾗﾙ'
  0008'    58 43 57 59
  000C'    36 45 7C 4C                      DB       "ｶﾅLITERAL"
  0010'    49 54 45 52
  0014'    41 4C
  0016'    7C 4C 49 7C                      DB       "LIﾃRAﾙ"
  001A'    43 7C 52 41
  001E'    7C 59
                                             END
```

3 参照)．実際にはこの位置に出てくるだけでなく，機械語コードに混じって出てきます．これらをすべて無視してソース・コード部分のみ変換するようにしなければなりません．

それではどうやって**.PRN** ファイルを変換するかが問題ですが，それには Pascal MT+を使用しました．詳細については第 6 章を参照してください．

4.4 マクロ命令のアセンブル時間

M80 でも全くマクロ命令を使わないコードをアセンブルすることはできます．一般には，マクロを使用すればプログラムを書く手間は省けますが，アセンブル時間は多くかかるといわれます．もし，全く同一のオブジェクト・プログラムを得るのにマクロを使用して作ったソース・ファイルと，マクロを使わないソース・ファイルをアセンブルした場合，時間としてはどの程度の差が出るのか具体的な資料がありません．

それで，実際に同一オブジェクトを得るプログラムを 2 種類作って試してみました．プログラムそのものは単純で AF, BC, DE, HL, IX, IY を **PUSH** し，その逆の順に **POP** する，これを 30 回繰り返すものです．マクロを使用する方は，図 3.19 のソーティング **PUSH** と **POP** です．これは通常考えられる最も複雑な構造を持ったマクロです．

これに対して，片や全く同じ内容をエディタで 360 行書いて作ったマクロを使わないものです．この両者のアセンブル時間を，M80 がロードされ終わってからエラーがないというメッセージが出るまでの値として計ってみました．その結果は，

　　マクロ使用　　　3 分 43 秒 (223 秒)

マクロ不使用　　　55 秒

となって約 4 倍の比になりました.

このマクロはソーティングという作業をやっていますから，他のマクロの数倍の時間がマクロだけにかかると考えると，他のマクロではほとんど差がないと思われます．差がない理由は，ソース・プログラムを読む時間が少なくなるからです．ところで，マクロを使った方は，

```
REPT    30
PUSHS   XYDHAB
POPS    XYDHAB
ENDM
```

とたった 4 行で書けます．マクロなしでは 360 行！この差を考えた時，4 倍のアセンブル時間は長いでしょうか，短いでしょうか？

テスト条件　：ハードウェア

PC 8001 mark II＋5 インチ 300 K バイトミニ FD×2

OS　　PC-CP/M 58 K

生成ファイル

.RELと.PRN

リスト制御

すべて無指定

4.5　サブルーチンの分割アセンブル例

図 4.10 はカナ文字メッセージを出すためのマクロなどをチェックするために作ったプログラムです．図 4.11 がインクルードされるマクロ定義とサブルーチンを含む別ファイルです．これをマクロ定義部とサブルーチンに分割して別アセンブルできるようにしたのが

〈図4.10〉カナ文字メッセージのチェック・プログラム

(a) ソース・リスト

```
 1:          .Z80
 2:  ;ｶﾅﾀｲｵｳ､ ｿﾌﾄﾀｲﾏ ﾉ ﾃｽﾄ ﾌﾟﾛｸﾞﾗﾑ
 3:  ;
 4:          INCLUDE B:B1.MAC  図4.11のファイルを取り込む
 5:  CRLFC:  DB      13,10,'$' ;ﾘﾀｰﾝ|ﾉ|ﾗｲﾝﾌｲｰﾄﾞ
 6:  PBN     EQU     800H+0EDH ;ﾋﾞｯﾄ|3,|ﾎﾟｰﾄ|0EDH
 7:  TMUNIT  EQU     50000 ;|50|ﾐﾘｾｺﾝﾄﾞ ﾀﾝｲ
 8:  TIMERN  EQU     2
 9:          DSEG
10:  TIMER0: DS      6*TIMERN
11:  BUF:    DS      128
12:  ECHOTM  EQU     TIMER0+6  ;ﾀｲﾏ|1|ﾆ ﾅﾏｴｦ ﾂｹﾙ
13:          CSEG
14:  START:  DI
15:          XOR     A
16:          OUT     (0E4H),A  ;|8214| ﾃﾞｾｰﾌﾞﾙ
17:          LD      (0EA55H),A
18:          CALL    INITM     ;ﾀｲﾏ ｲﾆｼｬﾙ
19:          CONJT   TIMER0,PBN,TIMER0,10
20:          TRIGT
21:          CONJT   ECHOTM,0,0,100
22:  LOOP:   MESAG   "ﾆｭｳﾘｮｸ ｼﾀ ﾓｼﾞ ｶﾞ ｼﾊﾞﾗｸﾀｯﾃ ｴｺｰ ｻﾚﾏｽ"
23:          MESAG   CRLFC
24:          MESAG   "ｶﾅ ﾄ |Alpha |ｦ ｺﾝｺﾞｳ ﾃﾞｷﾏｽ"
25:          MESAG   CRLFC
26:          LD      HL,BUF
27:          LD      B,128
28:  FILL:   LD      (HL),'$'  ;ｴﾝﾄﾞ ﾓｼﾞ ﾃﾞ ｳﾒﾙ
29:          INC     HL
30:          DJNZ    FILL
31:          LD      HL,125
32:          LD      (BUF),HL
33:          READL   BUF
34:          TRIGT   ECHOTM
35:          LD      A,-1
36:          OUT     (0E4H),A  ;|8214| ｲﾈｰﾌﾞﾙ
37:          EI
38:  MACHI:  LD      A,(ECHOTM)
39:          AND     A
40:          JR      NZ,MACHI
41:          DI
42:          XOR     A
43:          OUT     (0E4H),A  ;|8214| ﾃﾞｾｰﾌﾞﾙ
44:          LD      (0EA55H),A
45:          MESAG   CRLFC
46:          MESAG   CRLFC
```

```
47:            MESAG    BUF+2
48:            MESAG    CRLFC
49:            MESAG    BUF+2
50:            MESAG    CRLFC
51:            MESAG    CRLFC
52:            JP       LOOP       ;ｴﾝﾄﾞﾚｽ
53:  SWVECT:   LD       A,34H
54:            OUT      (0EBH),A   ;ﾀｲﾏ ﾃｲｼ
55:            JP       0          ;ｵﾜﾘ
56:            END      START
```

(b) アセンブル・リスト

```
                        .Z80
                ;ｶﾅﾀｲｵｳ､ ｿﾌﾄﾀｲﾏ ﾉ ﾃｽﾄ ﾌﾟﾛｸﾞﾗﾑ
                ;
        C               INCLUDE B:B1.MAC ← 図4.11
        C       ;ｲﾁﾄﾞ ﾉﾐ ﾃﾝｶｲｽﾙ ﾀﾒﾉ ﾁｪｯｸ
        C       ;     CHECK    ﾏｸﾛﾒｲ,EM
        C       ;        EM ﾊ ｻﾌﾞﾙｰﾁﾝ ﾄﾋﾞｺｼ ﾊﾞﾝﾁ
        C       ;          ｶﾅﾗｽﾞ LOCAL ﾆｽﾙ
        C       ;
        C       CHECK   MACRO   MNAME,EM
        C               IF      20H AND NOT TYPE MNAME
        C       MNAME   EQU     $+3
        C               ENDIF
        C               IF      MNAME EQ $+3
        C               JP      EM
        C               ENDM
        C       ;
        C       ;ｶﾅ ﾀｲｵｳ ﾒｯｾｰｼﾞ ﾏｸﾛ
        C       ;       MESAG   ﾊﾟﾗﾒｰﾀ
        C       ;       ﾊﾟﾗﾒｰﾀ ﾊ " ﾏﾀﾊ ' ﾃﾞ ｶｺﾏﾚﾃｲﾚﾊﾞ ﾘﾃﾗﾙ
        C       ;       ｿﾚｲｶﾞｲ ﾊ ｱﾄﾞﾚｽ(ｶﾝｾﾂ ｶﾉｳ)
        C       ;
        C       MESAG   MACRO   PARAM
        C               LOCAL   EM,TOBI,MES,J1,J2
        C               IRPC    X,PARAM
        C       CH      DEFL    '&X&X'
        C               EXITM
        C               ENDM
        C               IF      CH EQ 27H
        C               JR      TOBI
        C       MES:    DB      PARAM,'$'
        C       TOBI:   LD      HL,MES
        C               ELSE
        C               IF      CH EQ 2222H
        C               JR      TOBI
        C       MES:    DB      7CH,PARAM,'$'
        C       TOBI:   LD      HL,MES
        C               ELSE
        C               LD      HL,PARAM
        C               ENDIF
        C               ENDIF
        C               CALL    MESAG
        C               CHECK   MESAG,EM
        C               LD      BC,2
```

```
C       J1:     LD      A,(HL)
C               INC     HL
C               CP      '$'
C               RET     Z
C               CP      7CH
C               JR      NZ,J2
C               LD      A,80H
C               XOR     B
C               LD      B,A
C               JR      J1
C       J2:     OR      B
C               PUSH    HL
C               PUSH    BC
C               LD      E,A
C               CALL    5               ;CPM CALL
C               POP     BC
C               POP     HL
C               JR      J1
C       EM:
C               ENDIF
C               ENDM
C       ;
C       ;ｺﾝｿｰﾙ ｶﾗ 1 ｷﾞｮｳ ﾖﾐﾄﾙ
C       ;READL  ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
C       ;    ﾊﾞｯﾌｧ ｴﾘﾔ ﾊ ﾊﾞｯﾌｧｻｲｽﾞ ﾄ ﾓｼﾞｽｳ ﾉ
C       ;    2ﾊﾞｲﾄ ﾃﾞ ﾊｼﾞﾏﾙ
C       ;
C       READL   MACRO   ADRS
C               LD      DE,ADRS
C               LD      C,10
C               CALL    5
C               ENDM
C       ;
C       ;ｿﾌﾄｳｪｱ ﾀｲﾏ MACRO ﾃｲｷﾞ
C       ;ﾀｲﾏﾄﾘｶﾞTRIGT ﾀｲﾏﾒｲ､ﾀｲﾑﾁ､IOﾋﾞｯﾄ
C       ;   ﾀｲﾏﾒｲ ﾑｼﾃｲ ﾉﾄｷ TIMER0 ﾄｷﾒﾙ
C       ;   ﾀｲﾑﾁ ﾑｼﾃｲ ﾉﾄｷ ﾌﾟﾘｾｯﾄﾁ ｦ ｼﾖｳ
C       ;   IO ﾑｼﾃｲ ﾉﾄｷ ﾚﾝｹﾂ ﾓﾄﾉﾏﾏ
C       ;
C       TRIGT   MACRO   TIMER,CNT,PBN ;PBNﾉ ﾋﾞｯﾄ ｼﾃｲ ﾊ ﾋﾞｯﾄ ﾀｲｵｳ
C               IFB     <TIMER>
C               LD      A,(TIMER0+1)
C               LD      (TIMER0),A
C               ELSE
C               IFB     <CNT>
C               LD      A,(TIMER+1)
C               ELSE
C               LD      A,CNT
C               ENDIF
C               LD      (TIMER),A
C               ENDIF
C               IFNB    <PBN>
C               LD      HL,PBN
C               LD      (TIMER+2),HL
C               ENDIF
C               ENDM
```

```
                        C       ;
                        C       ;ﾀｲﾏｰ-IOﾚﾝｹﾂ
                        C       ;I CONJTI ﾀｲﾏﾒｲ､IOﾋﾞｯﾄ､ﾚﾝｹﾂﾀｲﾏﾒｲ､ﾌﾟﾘｾｯﾄﾀｲﾑﾁ
                        C       ;   ﾀｲﾏﾒｲ ﾊ ﾑｼﾃｲﾌｶ
                        C       ;   ｿﾉﾀ ﾊ ｼﾃｲﾉﾓﾉﾉﾐ ﾍﾝｺｳ ｽﾙ
                        C       ;
                        C       CONJT   MACRO   TIMER,PBN,TIMCJ,CNT
                        C               IFNB    <PBN>
                        C               LD      HL,PBN
                        C               LD      (TIMER+2),HL
                        C               ENDIF
                        C               IFNB    <TIMCJ>
                        C               LD      HL,TIMCJ
                        C               LD      (TIMER+4),HL
                        C               ENDIF
                        C               IFNB    <CNT>
                        C               LD      A,CNT
                        C               LD      (TIMER+1),A
                        C               ENDIF
                        C               ENDM
                        C       ;
                        C       ;ﾀｲﾏ ｶﾝｹｲ ｲﾆｼｬﾙ ﾙｰﾁﾝ
                        C       ;ﾀｲﾏ IC initialize (8253)
                        C       ;ｲﾝﾀﾗﾌﾟﾄ IC         (8214)
                        C       ;ﾊﾟﾗﾚﾙ IO          (8255) @ 0ECH
                        C       ;   ｷﾎﾝｼﾞｶﾝ ﾊ ﾏｲｸﾛSEC
                        C       ;
0000'   3E 34           C       INITM:  LD      A,34H ;ﾓｰﾄﾞ2
0002'   D3 EB           C               OUT     (0EBH),A ;8253ﾉ ﾎﾟｰﾄ
0004'   3E 50           C               LD      A,LOW TMUNIT
0006'   D3 E8           C               OUT     (0E8H),A
0008'   3E C3           C               LD      A,HIGH TMUNIT
000A'   D3 E8           C               OUT     (0E8H),A
000C'   3E EC           C               LD      A,0ECH  ;ﾍﾞｸﾀ ﾃｰﾌﾞﾙ ﾉ ﾍﾟｰｼﾞ
000E'   ED 47           C               LD      I,A
0010'   21 0021'        C               LD      HL,TMVECT
0013'   22 EC06         C               LD      (0EC06H),HL
0016'   21 016A'        C               LD      HL,SWVECT
0019'   22 EC0A         C               LD      (0EC0AH),HL
001C'   3E 90           C               LD      A,90H   ;ﾓｰﾄﾞ0 a,c:out b:in
001E'   D3 EF           C               OUT     (0EFH),A ;8255 ｺﾝﾄﾛｰﾙ
0020'   C9              C               RET
                        C       ;
                        C       ;ﾀｲﾏ ｲﾝﾀﾗﾌﾟﾄ ｼｮﾘ.  ﾀｲﾏ ﾉ ｶｽﾞTIMERN
                        C       ;                  ﾀｲﾏRAMｴﾘｱ TIMER0
                        C       ;         ﾚﾝﾄﾞｳ IO ﾎﾟｰﾄ 0 ﾊ ﾑｼｽﾙ
                        C       ;
0021'   08              C       TMVECT: EX      AF,AF'
0022'   D9              C               EXX
0023'   21 0000"        C               LD      HL,TIMER0
0026'   06 02           C               LD      B,TIMERN ;ﾀｲﾏ ﾉ ｶｽﾞ
0028'   11 0006         C       TMVE1:  LD      DE,6
002B'   3E FF           C               LD      A,-1
002D'   86              C               ADD     A,(HL)
002E'   30 27           C               JR      NC,TMVE5
0030'   77              C               LD      (HL),A
```

```
0031'   20 24          C             JR     NZ,TMVE5
0033'   23             C             INC    HL
0034'   23             C             INC    HL
0035'   4E             C             LD     C,(HL)   ;ﾚﾝﾄﾞｳ IO
0036'   0C             C             INC    C
0037'   0D             C             DEC    C
0038'   23             C             INC    HL
0039'   28 05          C             JR     Z,TMVE2
003B'   ED 78          C             IN     A,(C)
003D'   AE             C             XOR    (HL)
003E'   ED 79          C             OUT    (C),A
0040'   23             C     TMVE2:  INC    HL        ;ﾚﾝｹﾞﾂ ﾀｲﾏ
0041'   5E             C             LD     E,(HL)
0042'   23             C             INC    HL
0043'   56             C             LD     D,(HL)   ;ﾉ ｱﾄﾞﾚｽ DE
0044'   14             C             INC    D
0045'   15             C             DEC    D        ;0ﾍﾟｰｼﾞ ﾅﾗ ﾑｼ
0046'   28 04          C             JR     Z,TMVE3
0048'   13             C             INC    DE
0049'   1A             C             LD     A,(DE)   ;ﾀｲﾏ ﾌﾟﾘｾｯﾄﾁ ｦ
004A'   1B             C             DEC    DE
004B'   12             C             LD     (DE),A   ;ﾀｲﾏ ｽﾀｰﾄ
004C'   23             C     TMVE3:  INC    HL
004D'   10 D9          C             DJNZ   TMVE1
004F'   3E FF          C     TMVE4:  LD     A,-1
0051'   D3 E4          C             OUT    (0E4H),A ;8214 ｲﾈｰﾌﾞﾙ
0053'   D9             C             EXX
0054'   08             C             EX     AF,AF'
0055'   FB             C             EI
0056'   C9             C             RET
0057'   19             C     TMVE5:  ADD    HL,DE
0058'   10 D1          C             DJNZ   TMVE1+3
005A'   18 F3          C             JR     TMVE4
                       C     ;
005C'   0D 0A 24             CRLFC:  DB     13,10,'$' ;ﾘﾀﾝ/ﾗｲﾝﾌｲﾄﾞ
08ED                         PBN     EQU    800H+0EDH ;ﾋﾞｯﾄ3,ﾎﾟｰﾄ0EDH
C350                         TMUNIT  EQU    50000 ;50ﾐﾘｾｺﾝﾄﾞ ﾀﾝｲ
0002                         TIMERN  EQU    2
005F'                                DSEG
0000"                        TIMER0: DS     6*TIMERN
000C"                        BUF:    DS     128
0006"                        ECHOTM  EQU    TIMER0+6   ;ﾀｲﾏ1ﾆ ﾅﾏｴｦ ﾂｹﾙ
008C"                                CSEG
005F'   F3                   START:  DI
0060'   AF                           XOR    A
0061'   D3 E4                        OUT    (0E4H),A  ;8214 ﾃﾞｾｰﾌﾞﾙ
0063'   32 EA55                      LD     (0EA55H),A
0066'   CD 0000'                     CALL   INITM     ;ﾀｲﾏ ｲﾆｼｬﾙ
                                     CONJT  TIMER0,PBN,TIMER0,10
0069'   21 08ED        +             LD     HL,PBN
006C'   22 0002"       +             LD     (TIMER0+2),HL
006F'   21 0000"       +             LD     HL,TIMER0
0072'   22 0004"       +             LD     (TIMER0+4),HL
0075'   3E 0A          +             LD     A,10
0077'   32 0001"       +             LD     (TIMER0+1),A
                                     TRIGT
```

```
007A'   3A 0001"      +             LD      A,(TIMER0+1)
007D'   32 0000"      +             LD      (TIMER0),A
                                    CONJT   ECHOTM,0,0,100
0080'   21 0000       +             LD      HL,0
0083'   22 0008"      +             LD      (ECHOTM+2),HL
0086'   21 0000       +             LD      HL,0
0089'   22 000A"      +             LD      (ECHOTM+4),HL
008C'   3E 64         +             LD      A,100
008E'   32 0007"      +             LD      (ECHOTM+1),A
0091'                       LOOP:   MESAG   "ニュウリョク シタ モジ カ シハ゛ラクタッテ エコー サレマス"
0091'   18 24         +             JR      ..0001
0093'   7C 46 2D 33   +     ..0002: DB      7CH,"ニュウリョク シタ モジ カ シハ゛ラクタッテ エコー サレマス",
0097'   58 2E 38 20   +
009B'   3C 40 20 53   +
009F'   3C 5E 20 36   +
00A3'   5E 20 3C 4A   +
00A7'   5E 57 38 40   +
00AB'   2F 43 20 34   +
00AF'   3A 30 20 3B   +
00B3'   5A 4F 3D 24   +
00B7'   21 0093'      +     ..0001: LD      HL,..0002
00BA'   CD 00C0'      +             CALL    MESAG
00BD'   C3 00DD'      +             JP      ..0000
00C0'   01 0002       +             LD      BC,2
00C3'   7E            +     ..0003: LD      A,(HL)
00C4'   23            +             INC     HL
00C5'   FE 24         +             CP      '$'
00C7'   C8            +             RET     Z
00C8'   FE 7C         +             CP      7CH
00CA'   20 06         +             JR      NZ,..0004
00CC'   3E 80         +             LD      A,80H
00CE'   A8            +             XOR     B
00CF'   47            +             LD      B,A
00D0'   18 F1         +             JR      ..0003
00D2'   B0            +     ..0004: OR      B
00D3'   E5            +             PUSH    HL
00D4'   C5            +             PUSH    BC
00D5'   5F            +             LD      E,A
00D6'   CD 0005       +             CALL    5       ;CPM CALL
00D9'   C1            +             POP     BC
00DA'   E1            +             POP     HL
00DB'   18 E6         +             JR      ..0003
00DD'                 +     ..0000:
                                    MESAG   CRLFC
00DD'   21 005C'      +             LD      HL,CRLFC
00E0'   CD 00C0'      +             CALL    MESAG
                                    MESAG   "カナ ト Alpha ヲ コンゴ゛ウ デ゛キマス"
00E3'   18 1C         +             JR      ..000B
00E5'   7C 36 45 20   +     ..000C: DB      7CH,"カナ ト Alpha ヲ コンゴ゛ウ デ゛キマス",'$'
00E9'   44 20 7C 41   +
00ED'   6C 70 68 61   +
00F1'   20 7C 26 20   +
00F5'   3A 5D 3A 5E   +
00F9'   33 20 43 5E   +
00FD'   37 4F 3D 24   +
0101'   21 00E5'      +     ..000B: LD      HL,..000C
```

```
0104'   CD 00C0'     +           CALL   MESAG
                                 MESAG  CRLFC
0107'   21 005C'     +           LD     HL,CRLFC
010A'   CD 00C0'     +           CALL   MESAG
010D'   21 000C"                 LD     HL,BUF
0110'   06 80                    LD     B,128
0112'   36 24             FILL:  LD     (HL),'$'   ;ｴﾝﾄﾞ ﾓｼﾞ ﾃﾞ ｳﾒﾙ
0114'   23                       INC    HL
0115'   10 FB                    DJNZ   FILL
0117'   21 007D                  LD     HL,125
011A'   22 000C"                 LD     (BUF),HL
                                 READL  BUF
011D'   11 000C"     +           LD     DE,BUF
0120'   0E 0A        +           LD     C,10
0122'   CD 0005      +           CALL   5
                                 TRIGT  ECHOTM
0125'   3A 0007"     +           LD     A,(ECHOTM+1)
0128'   32 0006"     +           LD     (ECHOTM),A
012B'   3E FF                    LD     A,-1
012D'   D3 E4                    OUT    (0E4H),A   ;8214 ｲﾈｰﾌﾞﾙ
012F'   FB                       EI
0130'   3A 0006"          MACHI: LD     A,(ECHOTM)
0133'   A7                       AND    A
0134'   20 FA                    JR     NZ,MACHI
0136'   F3                       DI
0137'   AF                       XOR    A
0138'   D3 E4                    OUT    (0E4H),A   ;8214 ﾃﾞｾｰﾌﾞﾙ
013A'   32 EA55                  LD     (0EA55H),A
                                 MESAG  CRLFC
013D'   21 005C'     +           LD     HL,CRLFC
0140'   CD 00C0'     +           CALL   MESAG
                                 MESAG  CRLFC
0143'   21 005C'     +           LD     HL,CRLFC
0146'   CD 00C0'     +           CALL   MESAG
                                 MESAG  BUF+2
0149'   21 000E"     +           LD     HL,BUF+2
014C'   CD 00C0'     +           CALL   MESAG
                                 MESAG  CRLFC
014F'   21 005C'     +           LD     HL,CRLFC
0152'   CD 00C0'     +           CALL   MESAG
                                 MESAG  BUF+2
0155'   21 000E"     +           LD     HL,BUF+2
0158'   CD 00C0'     +           CALL   MESAG
                                 MESAG  CRLFC
015B'   21 005C'     +           LD     HL,CRLFC
015E'   CD 00C0'     +           CALL   MESAG
                                 MESAG  CRLFC
0161'   21 005C'     +           LD     HL,CRLFC
0164'   CD 00C0'     +           CALL   MESAG
0167'   C3 0091'                 JP     LOOP       ;ｴﾝﾄﾞﾚｽ
016A'   3E 34            SWVECT: LD     A,34H
016C'   D3 EB                    OUT    (0EBH),A   ;ﾀｲﾏ ﾃｲｼ
016E'   C3 0000                  JP     0          ;ｵﾜﾘ
                                 END    START
```

〈図 4.11〉インクルードされるマクロ定義とサブルーチンを含む別ファイル

```
 1:  ;ｲﾁﾄﾞ ﾉﾐ ﾃﾝｶｲｽﾙ ﾀﾒﾉ ﾁｪｯｸ
 2:  ;|      CHECK|      ﾏｸﾛﾒｲ|,EM
 3:  ;       |EM| ﾊ ｻﾌﾞﾙｰﾁﾝ ﾄﾋﾞｺｼ ﾊﾞﾝﾁ
 4:  ;         ｶﾅﾗｽﾞ  |LOCAL| ﾆｽﾙ
 5:  ;
 6:  CHECK     MACRO    MNAME,EM
 7:            IF       20H AND NOT TYPE MNAME
 8:  MNAME     EQU      $+3
 9:            ENDIF
10:            IF       MNAME EQ $+3
11:            JP       EM
12:            ENDM
13:  ;
14:  ;ｶﾅ ﾀｲｵｳ ﾒｯｾｰｼﾞ ﾏｸﾛ
15:  ;         |MESAG    |ﾊﾟﾗﾒｰﾀ
16:  ;         ﾊﾟﾗﾒｰﾀ ﾊ |"| ﾏﾀﾊ |'| ﾃﾞ ｶｺﾏﾚﾃｲﾚﾊﾞ ﾘﾃﾗﾙ
17:  ;         ｿﾚｲｶﾞｲ ﾊ ｱﾄﾞﾚｽ|(|ｶﾝｾﾂ ｶﾉｳ|)
18:  ;
19:  MESAG     MACRO    PARAM
20:            LOCAL    EM,TOBI,MES,J1,J2
21:            IRPC     X,PARAM
22:  CH        DEFL     '&X&X'
23:            EXITM
24:            ENDM
25:            IF       CH EQ 27H
26:            JR       TOBI
27:  MES:      DB       PARAM,'$'
28:  TOBI:     LD       HL,MES
29:            ELSE
30:            IF       CH EQ 2222H
31:            JR       TOBI
32:  MES:      DB       7CH,PARAM,'$'
33:  TOBI:     LD       HL,MES
34:            ELSE
35:            LD       HL,PARAM
36:            ENDIF
37:            ENDIF
38:            CALL     MESAG
39:            CHECK    MESAG,EM
40:            LD       BC,2
41:  J1:       LD       A,(HL)
42:            INC      HL
43:            CP       '$'
44:            RET      Z
45:            CP       7CH
46:            JR       NZ,J2
47:            LD       A,80H
48:            XOR      B
49:            LD       B,A
50:            JR       J1
```

このサブルーチンは
別にアセンブル可能
図4.14がその例

```
51:  J2:     OR      B
52:          PUSH    HL
53:          PUSH    BC
54:          LD      E,A
55:          CALL    5          ;｢CPM CALL
56:          POP     BC
57:          POP     HL
58:          JR      J1
59:  EM:
60:          ENDIF
61:          ENDM
62:  ;
63:  ;ｺﾝｿｰﾙ ｶﾗ ｢1｣ ｷﾞｮｳ ﾖﾐﾄﾙ
64:  ;｢READL｣  ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
65:  ;    ﾊﾞｯﾌｧ ｴﾘﾔ ﾊ ﾊﾞｯﾌｧｻｲｽﾞ ﾄ ﾓｼﾞｽｳ ﾉ
66:  ;｢    2｣ﾊﾞｲﾄ ﾃﾞ ﾊｼﾞﾏﾙ
67:  ;
68:  READL   MACRO   ADRS
69:          LD      DE,ADRS この行はマクロ・パラメータで変化するので別ファイル
                             にできない
70:          LD      C,10  ┐
71:          CALL    5     ┘別ファイル可能（図4.14）
72:          ENDM
73:  ;
74:  ;ｿﾌﾄｳｪｱ ﾀｲﾏ ｢MACRO｣ ﾃｲｷﾞ
75:  ;ﾀｲﾏﾄﾘｶﾞ｢TRIGT ｢ﾀｲﾏﾒｲ､ﾀｲﾑﾁ､｢IO｣ﾋﾞｯﾄ
76:  ;   ﾀｲﾏﾒｲ ﾑｼﾃｲ ﾉﾄｷ ｢TIMER0｣ ﾄｷﾒﾙ
77:  ;   ﾀｲﾑﾁ ﾑｼﾃｲ ﾉﾄｷ ﾌﾟﾘｾｯﾄﾁ ｦ ｼﾖｳ
78:  ;   ｢IO｣ ﾑｼﾃｲ ﾉﾄｷ ﾚﾝｹﾂ ﾓﾄﾉﾏﾏ
79:  ;
80:  TRIGT   MACRO   TIMER,CNT,PBN ;｢PBN｣ﾉ ﾋﾞｯﾄ ｼﾃｲ ﾊ ﾋﾞｯﾄ ﾀｲｵｳ
81:          IFB     <TIMER>
82:          LD      A,(TIMER0+1)
83:          LD      (TIMER0),A
84:          ELSE
85:          IFB     <CNT>
86:          LD      A,(TIMER+1)
87:          ELSE
88:          LD      A,CNT
89:          ENDIF
90:          LD      (TIMER),A
91:          ENDIF
92:          IFNB    <PBN>
93:          LD      HL,PBN
94:          LD      (TIMER+2),HL
95:          ENDIF
96:          ENDM
97:  ;
98:  ;ﾀｲﾏ｣--IO｣ﾚﾝｹﾂ
99:  ;         ｢CONJT｣ ﾀｲﾏﾒｲ､｢IO｣ﾋﾞｯﾄ､ﾚﾝｹﾂﾀｲﾏﾒｲ､ﾌﾟﾘｾｯﾄﾀｲﾑﾁ
```

```
100:   ;   ﾀｲﾏﾒｲ ﾊ ﾑｼﾃｲﾌｶ
101:   ;   ｿﾉﾀ ﾊ ｼﾃｲﾉﾓﾉﾉﾐ ﾍﾝｺｳ ｽﾙ
102:   ;
103:   CONJT      MACRO    TIMER,PBN,TIMCJ,CNT
104:              IFNB     <PBN>
105:              LD       HL,PBN
106:              LD       (TIMER+2),HL
107:              ENDIF       ,
108:              IFNB     <TIMCJ>
109:              LD       HL,TIMCJ
110:              LD       (TIMER+4),HL
111:              ENDIF
112:              IFNB     <CNT>
113:              LD       A,CNT
114:              LD       (TIMER+1),A
115:              ENDIF
116:              ENDM
117:   ;
118:   ;ﾀｲﾏ ｶﾝｹｲ ｲﾆｼｬﾙ ﾙｰﾁﾝ
119:   ;ﾀｲﾏ |IC initialize (8253)
120:   ;ｲﾝﾀﾗﾌﾟﾄ |IC      (8214)
121:   ;ﾊﾟﾗﾚﾙ |IO       (8255) @ 0ECH
122:   ;   ｷﾎﾝｼﾞｶﾝ ﾊ ﾏｲｸﾛ|SEC
123:   ;
124:   INITM:     LD       A,34H ;ﾓｰﾄﾞ|2
125:              OUT      (0EBH),A ;|8253|ﾉ ﾎﾟｰﾄ
126:              LD       A,LOW TMUNIT
127:              OUT      (0E8H),A
128:              LD       A,HIGH TMUNIT
129:              OUT      (0E8H),A
130:              LD       A,0ECH   ;ﾍﾞｸﾀ ﾃｰﾌﾞﾙ ﾉ ﾍﾟｰｼﾞ
131:              LD       I,A
132:              LD       HL,TMVECT
133:              LD       (0EC06H),HL
134:              LD       HL,SWVECT
135:              LD       (0EC0AH),HL
136:              LD       A,90H    ;ﾓｰﾄﾞ|0 a,c:out b:in
137:              OUT      (0EFH),A ;|8255| ｺﾝﾄﾛｰﾙ
138:              RET
139:   ;
140:   ;ﾀｲﾏ ｲﾝﾀﾗﾌﾟﾄ ｼｮﾘ。 ﾀｲﾏ ﾉ ｶｽﾞ|TIMERN
141:   ;                   ﾀｲﾏ|RAM|ｴﾘﾔ |TIMER0
142:   ;        ﾚﾝﾄﾞｳ |IO| ﾎﾟｰﾄ |0| ﾊ ﾑｼｽﾙ
143:   ;
144:   TMVECT:    EX       AF,AF'
145:              EXX
146:              LD       HL,TIMER0
147:              LD       B,TIMERN ;ﾀｲﾏ ﾉ ｶｽﾞ
148:   TMVE1:     LD       DE,6
149:              LD       A,-1
```

これ以後はすべてサブルーチンなので別ファイルにできる（図4.14）

```
150:            ADD     A,(HL)
151:            JR      NC,TMVE5
152:            LD      (HL),A
153:            JR      NZ,TMVE5
154:            INC     HL
155:            INC     HL
156:            LD      C,(HL)  ;ﾚﾝﾄﾞｳ !IO
157:            INC     C
158:            DEC     C
159:            INC     HL
160:            JR      Z,TMVE2
161:            IN      A,(C)
162:            XOR     (HL)
163:            OUT     (C),A
164:    TMVE2:  INC     HL      ;ﾚﾝｹﾞﾂ ﾀｲﾏ
165:            LD      E,(HL)
166:            INC     HL
167:            LD      D,(HL)  ;ﾉ ｱﾄﾞﾚｽ !DE
168:            INC     D
169:            DEC     D       ;!0!ﾍﾟｰｼﾞ ﾅﾗ ﾑｼ
170:            JR      Z,TMVE3
171:            INC     DE
172:            LD      A,(DE)  ;ﾀｲﾏ ﾌﾟﾘｾｯﾄﾁ ｦ
173:            DEC     DE
174:            LD      (DE),A  ;ﾀｲﾏ ｽﾀｰﾄ
175:    TMVE3:  INC     HL
176:            DJNZ    TMVE1
177:    TMVE4:  LD      A,-1
178:            OUT     (0E4H),A ;!8214 !ｲﾈｰﾌﾞﾙ
179:            EXX
180:            EX      AF,AF'
181:            EI
182:            RET
183:    TMVE5:  ADD     HL,DE
184:            DJNZ    TMVE1+3
185:            JR      TMVE4
186:    ;
```

〈図4.12〉図4.10を別アセンブル形式に変更

(a) ソース・リスト

```
 1:          .Z80
 2:  ;ｶﾅﾀｲｵｳ､  ｿﾌﾄﾀｲﾏ ﾉ ﾃｽﾄ ﾌﾟﾛｸﾞﾗﾑ
 3:  ;     ｷﾎﾝ ﾙｰﾁﾝ ｦ ﾍﾞﾂﾓｼﾞｭｰﾙ ﾄ ｽﾙ
 4:  ;
 5:          INCLUDE B:B3.MAC ←サブルーチンを含まないマクロ定義のみのファイル
                                   (図4.13)
 6:  ;
 7:          PUBLIC  TIMERN,TMUNIT,TIMER0,SWVECT  } 図4.10に追加した
 8:  ;                                            } のはこの2行だけ
 9:          EXT     INITM
10:
11:  CRLFC:  DB      13,10,'$'  ;ﾘﾀﾝ|/|ﾗｲﾝﾌｲﾄﾞ
12:  PBN     EQU     800H+0EDH ;ﾋﾞｯﾄ|3,|ﾎﾟｰﾄ|0EDH
13:  TMUNIT  EQU     50000 ;|50|ﾐﾘｾｺﾝﾄﾞ ﾀﾝｲ
```

（途中省略）

```
57:          MESAG   CRLFC
58:          JP      LOOP     ;ｴﾝﾄﾞﾚｽ
59:  SWVECT: LD      A,34H
60:          OUT     (0EBH),A   ;ﾀｲﾏ ﾃｲｼ
61:          JP      0        ;ｵﾜﾘ
62:          END     START
```

(b) アセンブル・リスト

```
                   .Z80
           ;ｶﾅﾀｲｵｳ､ ｿﾌﾄﾀｲﾏ ﾉ ﾃｽﾄ ﾌﾟﾛｸﾞﾗﾑ
           ;    ｷﾎﾝ ﾙｰﾁﾝ ｦ ﾍﾞﾂﾓｼﾞｭｰﾙ ﾄ ｽﾙ
           ;
   C               INCLUDE B:B3.MAC--------------(図4.13)
   C       ;
   C       ;ｶﾅ ﾀｲｵｳ ﾒｯｾｰｼﾞ ﾏｸﾛ
   C       ;       MESAG   ﾊﾟﾗﾒｰﾀ
   C       ;       ﾊﾟﾗﾒｰﾀ ﾊ " ﾏﾀﾊ ' ﾃﾞ ｶｺﾏﾚﾃｲﾚﾊﾞ ﾘﾃﾗﾙ
   C       ;       ｿﾚｲｶﾞｲ ﾊ ｱﾄﾞﾚｽ(ｶﾝｾﾂ ｶﾉｳ)
   C       ;
   C       MESAG   MACRO   PARAM
   C               LOCAL   TOBI,MES,J1,J2
   C               IRPC    X,PARAM
   C        CH     DEFL    '&X&X'
   C               EXITM
   C               ENDM
   C               IF      CH EQ 27H
   C               JR      TOBI
   C       MES:    DB      PARAM,'$'
   C       TOBI:   LD      HL,MES
   C               ELSE
   C               IF      CH EQ 2222H
   C               JR      TOBI
   C       MES:    DB      7CH,PARAM,'$'
   C       TOBI:   LD      HL,MES
```

```
C               ELSE
C               LD      HL,PARAM
C               ENDIF
C               ENDIF
C               CALL    MESAG##
C               ENDM
C       ;
C       ;ｺﾝｿｰﾙ ｶﾗ 1 ｷﾞｮｳ ﾖﾐﾄﾙ
C       ;READL ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
C       ;   ﾊﾞｯﾌｧ ｴﾘﾔ ﾊ ﾊﾞｯﾌｧｻｲｽﾞ ﾄ ﾓｼﾞｽｳ ﾉ
C       ;   2ﾊﾞｲﾄ ﾃﾞ ﾊｼﾞﾏﾙ
C       ;
C       READL   MACRO   ADRS
C               LD      DE,ADRS
C               CALL    READL##
C               ENDM
C       ;
C       ;ｿﾌﾄｳｪｱ ﾀｲﾏ MACRO ﾃｲｷﾞ
C       ;ﾀｲﾏﾄﾘｶﾞTRIGT ﾀｲﾏﾒｲ､ﾀｲﾑﾁ､IOﾋﾞｯﾄ
C       ;  ﾀｲﾏﾒｲ ﾑｼﾃｲ ﾉﾄｷ TIMER0 ﾄｷﾒﾙ
C       ;  ﾀｲﾑﾁ ﾑｼﾃｲ ﾉﾄｷ ﾌﾟﾘｾｯﾄﾁ ｦ ｼﾖｳ
C       ;  IO ﾑｼﾃｲ ﾉﾄｷ ﾚﾝｹﾂ ﾓﾄﾉﾏﾏ
C       ;
C       TRIGT   MACRO   TIMER,CNT,PBN ;PBNﾉ ﾋﾞｯﾄ ｼﾃｲ ﾊ ﾋﾞｯﾄ ﾀｲｵｩ
C               IFB     <TIMER>
C               LD      A,(TIMER0+1)
C               LD      (TIMER0),A
C               ELSE
C               IFB     <CNT>
C               LD      A,(TIMER+1)
C               ELSE
C               LD      A,CNT
C               ENDIF
C               LD      (TIMER),A
C               ENDIF
C               IFNB    <PBN>
C               LD      HL,PBN
C               LD      (TIMER+2),HL
C               ENDIF
C               ENDM
C       ;
C       ;ﾀｲﾏｰｰIOﾚﾝｹﾂ
C       ;|CONJT|ﾀｲﾏﾒｲ､IOﾋﾞｯﾄ､ﾚﾝｹﾂﾀｲﾏﾒｲ､ﾌﾟﾘｾｯﾄﾀｲﾑﾁ
C       ;  ﾀｲﾏﾒｲ ﾊ ﾑｼﾃｲﾌｶ
C       ;  ｿﾉﾀ ﾊ ｼﾃｲﾉﾓﾉﾉﾐ ﾍﾝｺｳ ｽﾙ
C       ;
C       CONJT   MACRO   TIMER,PBN,TIMCJ,CNT
C               IFNB    <PBN>
C               LD      HL,PBN
C               LD      (TIMER+2),HL
C               ENDIF
C               IFNB    <TIMCJ>
C               LD      HL,TIMCJ
C               LD      (TIMER+4),HL
C               ENDIF
C               IFNB    <CNT>
C               LD      A,CNT
C               LD      (TIMER+1),A
```

```
                   C                ENDIF
                   C                ENDM
                            ;
                                    PUBLIC   TIMERN,TMUNIT,TIMER0,SWVECT
                            ;
                                    EXT      INITM

0000'   0D 0A 24            CRLFC:  DB       13,10,'$'  ;ﾘﾀﾝ/ﾗｲﾝﾌｲｰﾄﾞ
08ED                        PBN     EQU      800H+0EDH ;ﾋﾞｯﾄ3,ﾎﾟｰﾄ0EDH
C350                        TMUNIT  EQU      50000 ;50ﾐﾘｾｺﾝﾄﾞ ﾀﾝｲ
0002                        TIMERN  EQU      2
0003'                               DSEG
0000"                       TIMER0: DS       6*TIMERN
000C"                       BUF:    DS       128
0006"                       ECHOTM  EQU      TIMER0+6   ;ﾀｲﾏ1ﾆ ﾅﾏｴｦ ﾂｹﾙ
008C"                               CSEG
0003'   F3                  START:  DI
0004'   AF                          XOR      A
0005'   D3 E4                       OUT      (0E4H),A   ;8214 ﾃﾞｾｰﾌﾞﾙ
0007'   32 EA55                     LD       (0EA55H),A
000A'   CD 0000*                    CALL     INITM     ;ﾀｲﾏ ｲﾆｼｬﾙ
                                    CONJT    TIMER0,PBN,TIMER0,10
000D'   21 08ED          +          LD       HL,PBN
0010'   22 0002"         +          LD       (TIMER0+2),HL
0013'   21 0000"         +          LD       HL,TIMER0
0016'   22 0004"         +          LD       (TIMER0+4),HL
0019'   3E 0A            +          LD       A,10
001B'   32 0001"         +          LD       (TIMER0+1),A
                                    TRIGT
001E'   3A 0001"         +          LD       A,(TIMER0+1)
0021'   32 0000"         +          LD       (TIMER0),A
                                    CONJT    ECHOTM,0,0,100
0024'   21 0000          +          LD       HL,0
0027'   22 0008"         +          LD       (ECHOTM+2),HL
002A'   21 0000          +          LD       HL,0
002D'   22 000A"         +          LD       (ECHOTM+4),HL
0030'   3E 64            +          LD       A,100
0032'   32 0007"         +          LD       (ECHOTM+1),A
0035'                       LOOP:   MESAG    "ﾆｭｳﾘｮｸ ｼﾀ ﾓｼﾞ ｶﾞ ｼﾊﾞﾗｸﾀｯﾃ ｴｺｰ ｻﾚﾏｽ"
0035'   18 24            +          JR       ..0000
0037'   7C 46 2D 33      +  ..0001: DB       7CH,"ﾆｭｳﾘｮｸ ｼﾀ ﾓｼﾞ ｶﾞ ｼﾊﾞﾗｸﾀｯﾃ ｴｺｰ ｻﾚﾏｽ",'$'
003B'   58 2E 38 20      +
003F'   3C 40 20 53      +
0043'   3C 5E 20 36      +
0047'   5E 20 3C 4A      +
004B'   5E 57 38 40      +
004F'   2F 43 20 34      +
0053'   3A 30 20 3B      +
0057'   5A 4F 3D 24      +
005B'   21 0037'         +  ..0000: LD       HL,..0001
005E'   CD 0000*         +          CALL     MESAG##
                                    MESAG    CRLFC
0061'   21 0000'         +          LD       HL,CRLFC
0064'   CD 0000*         +          CALL     MESAG##
                                    MESAG    "ｶﾅ ﾄ Alpha ｦ ｺﾝｺﾞｳ ﾃﾞｷﾏｽ"
0067'   18 1C            +          JR       ..0008
0069'   7C 36 45 20      +  ..0009: DB       7CH,"ｶﾅ ﾄ Alpha ｦ ｺﾝｺﾞｳ ﾃﾞｷﾏｽ",'$'
006D'   44 20 7C 41      +
0071'   6C 70 68 61      +
```

```
0075'   20 7C 26 20    +
0079'   3A 5D 3A 5E    +
007D'   33 20 43 5E    +
0081'   37 4F 3D 24    +
0085'   21 0069'       +        ..0008: LD      HL,..0009
0088'   CD 0000*       +                CALL    MESAG##
                                        MESAG   CRLFC
008B'   21 0000'       +                LD      HL,CRLFC
008E'   CD 0000*       +                CALL    MESAG##
0091'   21 000C"                        LD      HL,BUF
0094'   06 80                           LD      B,128
0096'   36 24                   FILL:   LD      (HL),'$'  ;ｴﾝﾄﾞ ﾓｼﾞ ﾃﾞ ｳﾒﾙ
0098'   23                              INC     HL
0099'   10 FB                           DJNZ    FILL
009B'   21 007D                         LD      HL,125
009E'   22 000C"                        LD      (BUF),HL
                                        READL   BUF
00A1'   11 000C"       +                LD      DE,BUF
00A4'   CD 0000*       +                CALL    READL##
                                        TRIGT   ECHOTM
00A7'   3A 0007"       +                LD      A,(ECHOTM+1)
00AA'   32 0006"       +                LD      (ECHOTM),A
00AD'   3E FF                           LD      A,-1
00AF'   D3 E4                           OUT     (0E4H),A  ;8214 ｲﾈｰﾌﾞﾙ
00B1'   FB                              EI
00B2'   3A 0006"                MACHI:  LD      A,(ECHOTM)
00B5'   A7                              AND     A
00B6'   20 FA                           JR      NZ,MACHI
00B8'   F3                              DI
00B9'   AF                              XOR     A
00BA'   D3 E4                           OUT     (0E4H),A  ;8214 ﾃﾞｾｰﾌﾞﾙ
00BC'   32 EA55                         LD      (0EA55H),A
                                        MESAG   CRLFC
00BF'   21 0000'       +                LD      HL,CRLFC
00C2'   CD 0000*       +                CALL    MESAG##
                                        MESAG   CRLFC
00C5'   21 0000'       +                LD      HL,CRLFC
00C8'   CD 0000*       +                CALL    MESAG##
                                        MESAG   BUF+2
00CB'   21 000E"       +                LD      HL,BUF+2
00CE'   CD 0000*       +                CALL    MESAG##
                                        MESAG   CRLFC
00D1'   21 0000'       +                LD      HL,CRLFC
00D4'   CD 0000*       +                CALL    MESAG##
                                        MESAG   BUF+2
00D7'   21 000E"       +                LD      HL,BUF+2
00DA'   CD 0000*       +                CALL    MESAG##
                                        MESAG   CRLFC
00DD'   21 0000'       +                LD      HL,CRLFC
00E0'   CD 0000*       +                CALL    MESAG##
                                        MESAG   CRLFC
00E3'   21 0000'       +                LD      HL,CRLFC
00E6'   CD 0000*       +                CALL    MESAG##
00E9'   C3 0035'                        JP      LOOP      ;ｴﾝﾄﾞﾚｽ
00EC'   3E 34                   SWVECT: LD      A,34H
00EE'   D3 EB                           OUT     (0EBH),A  ;ﾀｲﾏ ﾃｲｼ
00F0'   C3 0000                         JP      0         ;ｵﾜﾘ
                                        END     START
```

図4.13と図4.14です．図4.10もグローバル宣言をする2行を先頭に追加して図4.12のようになっています．

ここにひとつノウハウがあります．すでに完成してデバッグ済みのプログラムを分割する場合，どのシンボルが内部で定義されていて，どれが外部から見えなくてはならないか……と考えるのは面倒です．そこでいきなり分割だけしてアセンブルします．これは当然エラーが続出します．コンソールに次々とエラー表示が出ます．このエラーになったシンボルがすべて **EXTERNAL** に宣言しなければならないシンボルです．では，コンソールに出てくるエラー表示をメモしたり，プリントしたりしなければならないかといえば，その必要はありません．大切なのはリストを取っておくことです．さほど大きくないプログラムならリストをコンソールに出しておけば最後にシンボル表が出てきます．この中に **U** の記号がついているシンボルが内部で決まらないものです．

では，エラー・メッセージとリストの最後のシンボル表とはどこが違うのでしょうか．それはエラーの数とシンボル表の **U** の数が違うのです．エラーは未定義のシンボルを参照しようとするたびに出てきます．これに対してシンボル表では同一の未定義シンボルを何回も表示することはありません．つまり，シンボル表の **U** の数が即 **EXTERNAL** に宣言するべきシンボルの数というわけです．

エラー表示で調べると同じシンボルが何回も出てきてしまい，重複をさけるのが大変です．しかもエラー表示は1行に1エラーですから，エラー23個でスクロール・アウトします．シンボル表は最後の16行が残ります．もし，シンボルの数が多くてコンソールでは見きれない場合は，リストを一度ファイルに取っておいて，そのファイルを **PIP** でプリンタへ出力します．このとき，

```
                          ┌── Ctl-Z
A>PIP ⏎                   ↓
* LST：＝ファイル名.PRN [SSym^Z] ⏎
```

と入力します．これでシンボル表と最後のエラー・メッセージがプリントされます．

シンボル表の **U** をさがして **EXTERNAL** にしたものは，別のファイルで **PUBLIC** にしなければなりません．ファイルが多ければ，どのファイルで定義されているべきものか調べるのは大変ですが，ファイルが2つなら相手はひとつだけですからお互いにこちらで **EXTERNAL** にしたら相手側で **PUBLIC** にすればよいわけです．

相手が多い時はシンボル表をプリントしてにらめっこします．シンボル表はソートされていますから簡単に見付かります．

〈図 4.13〉 図 4.11 からマクロ定義部分のみを取り出す

```
  1:  ;
  2:  ;ｶﾅ ﾀｲｵｳ ﾒｯｾｰｼﾞ ﾏｸﾛ
  3:  ;          |MESAG    |ﾊﾟﾗﾒｰﾀ
  4:  ;          ﾊﾟﾗﾒｰﾀ ﾊ |"| ﾏﾀﾊ |'| ﾃﾞ ｶｺﾏﾚﾃｲﾚﾊﾞ ﾘﾃﾗﾙ
  5:  ;          ｿﾚｲｶﾞｲ ﾊ ｱﾄﾞﾚｽ|(|ｶﾝｾﾂ ｶﾉｳ|)
  6:  ;
  7:  MESAG     MACRO     PARAM
  8:            LOCAL     TOBI,MES,J1,J2
  9:            IRPC      X,PARAM
 10:  CH        DEFL      '&X&X'
 11:            EXITM
 12:            ENDM
 13:            IF        CH EQ 27H
 14:            JR        TOBI
 15:  MES:      DB        PARAM,'$'
 16:  TOBI:     LD        HL,MES
 17:            ELSE
 18:            IF        CH EQ 2222H
 19:            JR        TOBI
 20:  MES:      DB        7CH,PARAM,'$'
 21:  TOBI:     LD        HL,MES
 22:            ELSE
 23:            LD        HL,PARAM
 24:            ENDIF
 25:            ENDIF
 26:            CALL      MESAG##
 27:            ENDM
 28:  ;
 29:  ;ｺﾝｿｰﾙ ｶﾗ |1| ｷﾞｮｳ ﾖﾐﾄﾙ
 30:  ;|READL|   ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
 31:  ;    ﾊﾞｯﾌｧ ｴﾘﾔ ﾊ ﾊﾞｯﾌｧｻｲｽﾞ ﾄ ﾓｼﾞｽｳ ﾉ
 32:  ;|    2|ﾊﾞｲﾄ ﾃﾞ ﾊｼﾞﾏﾙ
 33:  ;
 34:  READL     MACRO     ADRS
 35:            LD        DE,ADRS
 36:            CALL      READL##
 37:            ENDM
 38:  ;
 39:  ;ｿﾌﾄｳｪｱ ﾀｲﾏ |MACRO| ﾃｲｷﾞ
 40:  ;ﾀｲﾏﾄﾘｶﾞ|TRIGT |ﾀｲﾏﾒｲ､ﾀｲﾑﾁ､|IO|ﾋﾞｯﾄ
 41:  ;   ﾀｲﾏﾒｲ ﾑｼﾃｲ ﾉﾄｷ |TIMER0| ﾄｷﾒﾙ
 42:  ;   ﾀｲﾑﾁ ﾑｼﾃｲ ﾉﾄｷ ﾌﾟﾘｾｯﾄﾁ ｦ ｼﾖｳ
 43:  ;   |IO| ﾑｼﾃｲ ﾉﾄｷ ﾚﾝｹﾂ ﾓﾄﾉﾏﾏ
 44:  ;
 45:  TRIGT     MACRO     TIMER,CNT,PBN ;|PBN|ﾉ ﾋﾞｯﾄ ｼﾃｲ ﾊ ﾋﾞｯﾄ ﾀｲｵｳ
 46:            IFB       <TIMER>
 47:            LD        A,(TIMER0+1)
 48:            LD        (TIMER0),A
```

マクロ処理を別モジュールとしたので外部に求める指示をする

```
49:            ELSE
50:            IFB       <CNT>
51:            LD        A,(TIMER+1)
52:            ELSE
53:            LD        A,CNT
54:            ENDIF
55:            LD        (TIMER),A
56:            ENDIF
57:            IFNB      <PBN>
58:            LD        HL,PBN
59:            LD        (TIMER+2),HL
60:            ENDIF
61:            ENDM
62:  ;
63:  ;ﾀｲﾏ!--IO!ﾚﾝｹﾂ
64:  ;         !CONJT! ﾀｲﾏﾒｲ､!IO!ﾋﾞｯﾄ､ﾚﾝｹﾂﾀｲﾏﾒｲ､ﾌﾟﾘｾｯﾄﾀｲﾑﾁ
65:  ;   ﾀｲﾏﾒｲ ﾊ ﾑｼﾃｲﾌｶ
66:  ;   ｿﾉﾀ ﾊ ｼﾃｲﾉﾓﾉﾉﾐ ﾍﾝｺｳ ｽﾙ
67:  ;
68:  CONJT     MACRO     TIMER,PBN,TIMCJ,CNT
69:            IFNB      <PBN>
70:            LD        HL,PBN
71:            LD        (TIMER+2),HL
72:            ENDIF
73:            IFNB      <TIMCJ>
74:            LD        HL,TIMCJ
75:            LD        (TIMER+4),HL
76:            ENDIF
77:            IFNB      <CNT>
78:            LD        A,CNT
79:            LD        (TIMER+1),A
80:            ENDIF
81:            ENDM
```

〈図 4.14〉 別アセンブルされるマクロ処理部

(a) ソース・リスト

```
 1:            .Z80      ;
 2:            NAME      ('MESTIM') ;ﾒｯｾｰｼﾞ ﾄ ﾀｲﾏ
 3:  ;
 4:  ;ｶﾅ ﾀｲｵｳ ﾒｯｾｰｼﾞ ﾏｸﾛ
 5:  ;   ｼｮﾘ ﾙｰﾁﾝ
 6:  ;
 7:  MESAG::   LD        BC,2
 8:  J1:       LD        A,(HL)
 9:            INC       HL
10:            CP        '$'
11:            RET       Z
```

```
12:              CP      7CH
13:              JR      NZ,J2
14:              LD      A,80H
15:              XOR     B
16:              LD      B,A
17:              JR      J1
18:  J2:         OR      B
19:              PUSH    HL
20:              PUSH    BC
21:              LD      E,A
22:              CALL    5          ;¦CPM CALL
23:              POP     BC
24:              POP     HL
25:              JR      J1
26:  ;
27:  ;ｺﾝｿｰﾙ ｶﾗ ¦1¦ ｷﾞｮｳ ﾖﾐﾄﾙ
28:  ;
29:  READL::     LD      C,10
30:              CALL    5
31:              RET
32:  ;
33:  ;ﾀｲﾏ ｶﾝｹｲ ｲﾆｼｬﾙ ﾙｰﾁﾝ
34:  ;ﾀｲﾏ ¦IC initialize (8253)
35:  ;ｲﾝﾀﾗﾌﾟﾄ ¦IC      (8214)
36:  ;ﾊﾟﾗﾚﾙ ¦IO       (8255) @ 0ECH
37:  ;   ｷﾎﾝｼﾞｶﾝ ﾊ ﾏｲｸﾛ¦SEC
38:  ;
39:  INITM::     LD      A,34H ;ﾓｰﾄﾞ¦2
40:              OUT     (0EBH),A ;¦8253¦ﾉ ﾎﾟｰﾄ
41:              LD      A,LOW TMUNIT##
42:              OUT     (0E8H),A
43:              LD      A,HIGH TMUNIT##
44:              OUT     (0E8H),A
45:              LD      A,0ECH  ;ﾍﾞｸﾀ ﾃｰﾌﾞﾙ ﾉ ﾍﾟｰｼﾞ
46:              LD      I,A
47:              LD      HL,TMVECT
48:              LD      (0EC06H),HL
49:              LD      HL,SWVECT##
50:              LD      (0EC0AH),HL
51:              LD      A,90H  ;ﾓｰﾄﾞ¦0 a,c:out b:in
52:              OUT     (0EFH),A ;¦8255¦ ｺﾝﾄﾛｰﾙ
53:              RET
54:  ;
55:  ;ﾀｲﾏ ｲﾝﾀﾗﾌﾟﾄ ｼｮﾘ｡ ﾀｲﾏ ﾉ ｶｽﾞ¦TIMERN
56:  ;                  ﾀｲﾏ¦RAM¦ｴﾘﾔ ¦TIMER0
57:  ;       ﾚﾝﾄﾞｳ ¦IO¦ ﾎﾟｰﾄ ¦0¦ ﾊ ﾑｼｽﾙ
58:  ;
```

マクロショリがメイン・プログラムから見えるようにする

メイン・プログラムで定義される

```
 59:  TMVECT: EX      AF,AF'
 60:          EXX
 61:          LD      HL,TIMER0##
 62:          LD      B,TIMERN##  ;ﾀｲﾏ ﾉ ｶｽﾞ
 63:  TMVE1:  LD      DE,6
 64:          LD      A,-1
 65:          ADD     A,(HL)
 66:          JR      NC,TMVE5
 67:          LD      (HL),A
 68:          JR      NZ,TMVE5
 69:          INC     HL
 70:          INC     HL
 71:          LD      C,(HL)   ;ﾚﾝﾄﾞｳ !IO
 72:          INC     C
 73:          DEC     C
 74:          INC     HL
 75:          JR      Z,TMVE2
 76:          IN      A,(C)
 77:          XOR     (HL)
 78:          OUT     (C),A
 79:  TMVE2:  INC     HL          ;ﾚﾝｹﾂ ﾀｲﾏ
 80:          LD      E,(HL)
 81:          INC     HL
 82:          LD      D,(HL)   ;ﾉ ｱﾄﾞﾚｽ !DE
 83:          INC     D
 84:          DEC     D           ;!0!ﾍﾟｰｼﾞ ﾅﾗ ﾑｼ
 85:          JR      Z,TMVE3
 86:          INC     DE
 87:          LD      A,(DE)   ;ﾀｲﾏ ﾌﾟﾘｾｯﾄﾁ ｦ
 88:          DEC     DE
 89:          LD      (DE),A   ;ﾀｲﾏ ｽﾀｰﾄ
 90:  TMVE3:  INC     HL
 91:          DJNZ    TMVE1
 92:  TMVE4:  LD      A,-1
 93:          OUT     (0E4H),A ;!8214 !ｲﾈｰﾌﾞﾙ
 94:          EXX
 95:          EX      AF,AF'
 96:          EI
 97:          RET
 98:  TMVE5:  ADD     HL,DE
 99:          DJNZ    TMVE1+3
100:          JR      TMVE4
101:  ;
102:          END         ;ﾓｼﾞｭｰﾙ ｴﾝﾄﾞ
```

メイン・プログラムで定義される

(b) アセンブル・リスト

```
                                 .Z80       ;
                                 NAME       ('MESTIM') ;ﾒｯｾｰｼﾞ ﾄ ﾀｲﾏ
                        ;
                        ;ｶﾅ ﾀｲｵｳ ﾒﾂｾｰｼﾞ ﾏｸﾛ
                        ;   ｼｮﾘ ﾙｰﾁﾝ
                        ;
0000'   01 0002         MESAG:: LD         BC,2
0003'   7E              J1:     LD         A,(HL)
0004'   23                      INC        HL
0005'   FE 24                   CP         '$'
0007'   C8                      RET        Z
0008'   FE 7C                   CP         7CH
000A'   20 06                   JR         NZ,J2
000C'   3E 80                   LD         A,80H
000E'   A8                      XOR        B
000F'   47                      LD         B,A
0010'   18 F1                   JR         J1
0012'   B0              J2:     OR         B
0013'   E5                      PUSH       HL
0014'   C5                      PUSH       BC
0015'   5F                      LD         E,A
0016'   CD 0005                 CALL       5          ;CPM CALL
0019'   C1                      POP        BC
001A'   E1                      POP        HL
001B'   18 E6                   JR         J1
                        ;
                        ;ｺﾝｿｰﾙ ｶﾗ 1 ｷﾞｮｳ ﾖﾐﾄﾙ
                        ;
001D'   0E 0A           READL:: LD         C,10
001F'   CD 0005                 CALL       5
0022'   C9                      RET
                        ;
                        ;ﾀｲﾏ ｶﾝｹｲ ｲﾆｼｬﾙ ﾙｰﾁﾝ
                        ;ﾀｲﾏ IC initialize (8253)
                        ;ｲﾝﾀﾗﾌﾟﾄ IC         (8214)
                        ;ﾊﾟﾗﾚﾙ IO           (8255) @ 0ECH
                        ;   ｷﾎﾝｼﾞｶﾝ ﾊ ﾏｲｸﾛSEC
                        ;
0023'   3E 34           INITM:: LD         A,34H ;ﾓｰﾄﾞ2
0025'   D3 EB                   OUT        (0EBH),A ;8253ﾉ ﾎﾟｰﾄ
0027'   3E 00*                  LD         A,LOW TMUNIT##
0029'   D3 E8                   OUT        (0E8H),A
002B'   3E 00*                  LD         A,HIGH TMUNIT##
002D'   D3 E8                   OUT        (0E8H),A
002F'   3E EC                   LD         A,0ECH  ;ﾍﾞｸﾀ ﾃｰﾌﾞﾙ ﾉ ﾍﾟｰｼﾞ
0031'   ED 47                   LD         I,A
0033'   21 0044'                LD         HL,TMVECT
0036'   22 EC06                 LD         (0EC06H),HL
0039'   21 0000*                LD         HL,SWVECT##
003C'   22 EC0A                 LD         (0EC0AH),HL
003F'   3E 90                   LD         A,90H  ;ﾓｰﾄﾞ0 a,c:out b:in
0041'   D3 EF                   OUT        (0EFH),A ;8255 ｺﾝﾄﾛｰﾙ
0043'   C9                      RET
```

```
                                        ;
                                        ;ﾀｲﾏ ｲﾝﾀﾗﾌﾟﾄ ｼｮﾘ｡ ﾀｲﾏ ﾉ ｶｽﾞTIMERN
                                        ;                ﾀｲﾏRAMｴﾘﾔ TIMER0
                                        ;        ﾚﾝﾄﾞｳ IO ﾎﾟｰﾄ 0 ﾊ ﾑｼｽﾙ
                                        ;
0044'   08                              TMVECT: EX      AF,AF'
0045'   D9                                      EXX
0046'   21 0000*                                LD      HL,TIMER0##
0049'   06 00*                                  LD      B,TIMERN## ;ﾀｲﾏ ﾉ ｶｽﾞ
004B'   11 0006                         TMVE1:  LD      DE,6
004E'   [illegible] FF                          LD      A,-1
0050'   [illegible]                             ADD     A,(HL)
0051'   30 27                                   JR      NC,TMVE5
0053'   77                                      LD      (HL),A
0054'   20 24                                   JR      NZ,TMVE5
0056'   23                                      INC     HL
0057'   23                                      INC     HL
0058'   4E                                      LD      C,(HL)   ;ﾚﾝﾄﾞｳ IO
0059'   0C                                      INC     C
005A'   0D                                      DEC     C
005B'   23                                      INC     HL
005C'   28 05                                   JR      Z,TMVE2
005E'   ED 78                                   IN      A,(C)
0060'   AE                                      XOR     (HL)
0061'   ED 79                                   OUT     (C),A
0063'   23                              TMVE2:  INC     HL       ;ﾚﾝｹﾂ ﾀｲﾏ
0064'   5E                                      LD      E,(HL)
0065'   23                                      INC     HL
0066'   56                                      LD      D,(HL)   ;ﾉ ｱﾄﾞﾚｽ DE
0067'   14                                      INC     D
0068'   15                                      DEC     D        ;0ﾍﾟｰｼﾞ ﾅﾗ ﾑｼ
0069'   28 04                                   JR      Z,TMVE3
006B'   13                                      INC     DE
006C'   1A                                      LD      A,(DE)   ;ﾀｲﾏ ﾌﾟﾘｾｯﾄﾁ ｦ
006D'   1B                                      DEC     DE
006E'   12                                      LD      (DE),A   ;ﾀｲﾏ ｽﾀｰﾄ
006F'   23                              TMVE3:  INC     HL
0070'   10 D9                                   DJNZ    TMVE1
0072'   3E FF                           TMVE4:  LD      A,-1
0074'   D3 E4                                   OUT     (0E4H),A ;8214 ｲﾈｰﾌﾞﾙ
0076'   D9                                      EXX
0077'   08                                      EX      AF,AF'
0078'   FB                                      EI
0079'   C9                                      RET
007A'   19                              TMVE5:  ADD     HL,DE
007B'   10 D1                                   DJNZ    TMVE1+3
007D'   18 F3                                   JR      TMVE4
                                        ;
                                                END     ;ﾓｼﾞｭｰﾙ ｴﾝﾄﾞ
```

4.6 実行時リロケート機能 ——ROM内のプログラムをRAMに移して実行

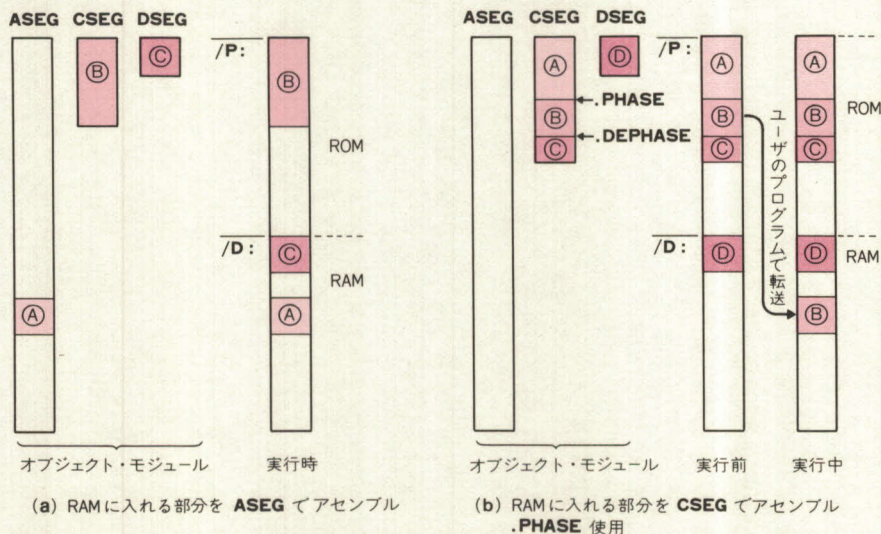
プログラムをRAMに移して実行したい理由はいろいろ考えられます．たとえば，**BIT**命令のビット番号を変更したいとか，ジャンプ命令の飛び先を書き変えたいなど，それにRAMの方がスピードが速いというハード上の条件もあり得ます．RAMに移す部分が数ステップなら簡単ですが，これが大きくなるとアセンブラで番地を付けないと大変な手間がかかります．

RAMに入れる番地が固定されているものとすれば，これを**ASEG**としてアセンブルすると図4.15(a)のようなメモリ配置が得られます．もちろん，**CSEG**の中から**ASEG**を参照したり，その逆もすべてアセンブラとリンカが解決してくれます．しかし，このコードをそのままROMに書いても**ASEG**の番地はそのままでは書けません．RAMの番地がROMから離れているからです．

そこで，通常はオペレータの責任でこの部分を取り出してROMのある場所に書き，**CSEG**のプログラムでその場所をあらかじめ指定しておいてRAMに移す作業をしなけ

〈図4.15〉RAM中で実行されるプログラム

(a) RAMに入れる部分を **ASEG** でアセンブル

(b) RAMに入れる部分を **CSEG** でアセンブル **.PHASE** 使用

ればなりません．ここで問題なのは，**ASEG** の部分を ROM のどこへ書き込むかが，ROM 書き込み作業段階で決定できる点です．プログラムを書いた本人がこの作業をすればよいとしても，分業されていると混乱する可能性は大です．

このようなトラブルをさけるために，M80 では実行時に別のエリアに移して実行されるコードの番地を解決するための疑似命令が用意されています．それは，**.PHASE** と **.DEPHASE** です．この機能を使用しても ROM から RAM へ移すプログラムはユーザの手で書かなければなりません．しかし，この部分を ROM のどの場所に書いておくか，ROM が何バイト必要になるかなどの番地に関しては，すべて **CSEG** と同じ扱いになっていますから，リンク作業で他のプログラムと結合されても，また ROM 内の番地を変更しても RAM へ移して実行する上で問題は起きません．

ユーザは，**.PHASE～.DEPHASE** の部分が **CSEG** の中の何処に位置するかについて考える必要はなく，すべてラベルを使って RAM へ転送するプログラムを作っておけばよいのです．図 4.15(b)がこの方法による場合のメモリ配置です．**.PHASE** の使用例は 6.4.1 に出てきます．

第 5 章
マクロ・アセンブラの高度な応用

M80 程度のマクロ機能があるとその応用可能性はほぼ無限といってもよく，とうてい本書だけで説明しきれません．マクロ機能だけをとらえてもそうですから，まして機械語プログラムの技術と組み合わせて新たな可能性を追求して行くと，あまりの奥の深さに圧倒されそうです．第 2 章，第 3 章で呼び鈴を押して，第 4 章で門が開き，本章で初めて長いアプローチに入るといっても過言ではありません．

前章までは基本事項の断片の集合であったともいえます．本章では，実際の使用により即した応用例をとり上げたいと思います．

まず最初に，図 4.10～図 4.14 のプログラムとソフトタイマについて．つぎに算術演算の同一マクロ呼び出しに対する種々の展開法．そして，同じく算術演算の高級言語風表記法について応用例を紹介します．高級言語風の表記法は数値演算の多い大規模なプログラムの開発では抜群の威力を発揮します．また，インタプリタ式のマクロ展開ではメモリを極端に節約することもできます．

後半ではビット・プロセッサ（1 ビット単位のデータを処理するプロセッサ）用のクロス・アセンブラと Z80 用ロジック・エミュレータ(Z80 でロジック処理をビット単位で実行する)の論理式表記．BPU 用クロス・コンパイラ風表記について解説します．

5.1 カナ文字対応メッセージ出力

図 4.10(a)で 22 行目と 24 行目にカナ文字メッセージのマクロ呼び出しが書かれています．通常の処理でこのカナ文字が出力できないので，図 4.11 に示すマクロ定義を使用してカナとアルファ数字に対応できるようにしたものです．このマクロの展開フローは図 5.1 に，また処理ルーチン(これもマクロの中に含まれているが一度のみしか展開されない)は図 5.2 のようなフローにしたがっています．

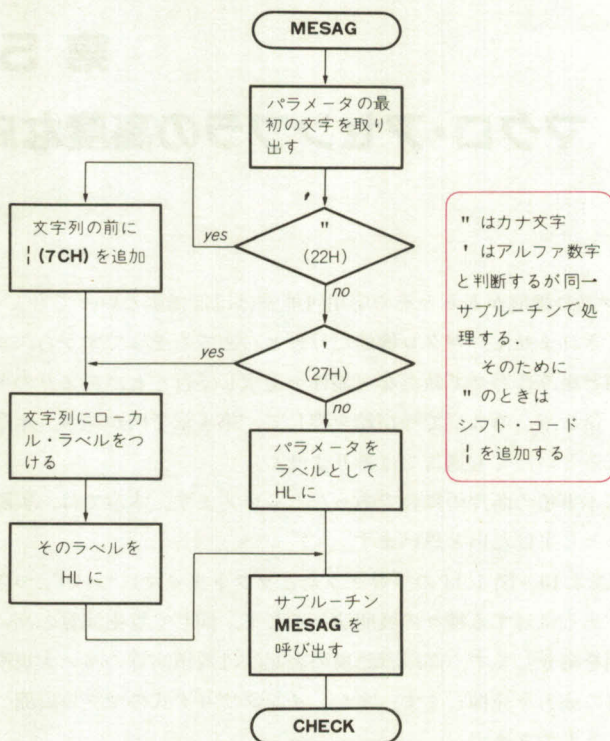
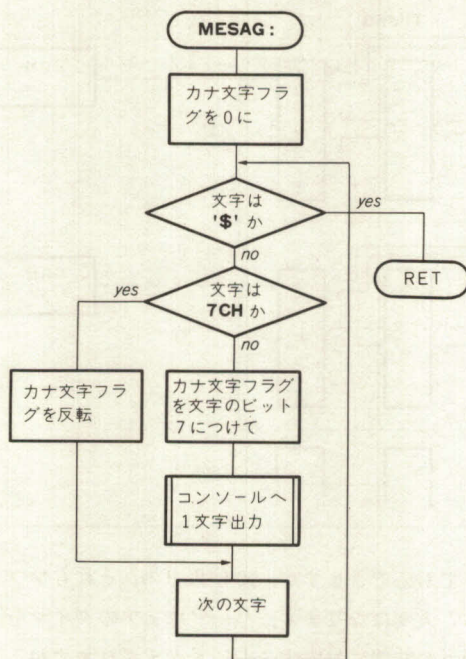
〈図 5.1〉 カナ文字対応メッセージ・マクロの展開フロー

引用符が〝'〟であったか〝"〟であったかの情報は，何もしなければオブジェクト・コード上には反映されませんから，マクロ展開時に〝"〟であった時は **7CH** を文字列の前につけて処理データとします. 当然, このマクロを通して **7CH** の文字そのものを出力することはできません. **$**を終わりの記号としているのは，カナ対応でない時のシステム・コール番号 9 のなごりです．このサブルーチンはアルファ数字の状態から始まります．したがって，直接システム・コール 9 を使っていたマクロと互換性があり，以前のマクロ呼び出しを全く変更する必要がありません.

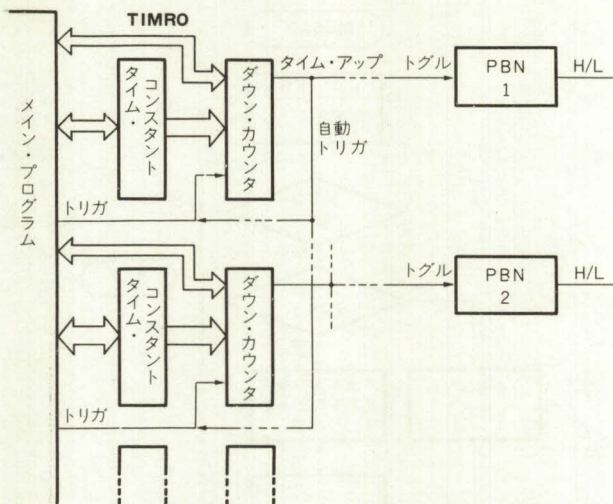
図 4.10 (b)がアセンブル・リストです．メッセージの 16 進コードは確かにビット 7 が 0 になっていますが，ソース部分のリストはカナ文字に復元されています(4.3 節参照).

〈図 5.2〉カナ文字対応メッセージ・サブルーチン実行フロー

このソフトは読み取った文字列を，時間を待ってからエコーするようになっていますが，エコー用にも同じ **MESAG** を使用していますから，入力された文字の中に **7CH** のコードが入っているとその文字が消えるだけでなく，それ以後の文字が変わってしまいます．実用上は **7CH** を使用する必要性をチェックしておかなければなりません．

5.2 ソフトウェアによるマルチ・タイマ

リアルタイムで動作するソフトウェアにとって，タイマ機能は不可欠です．アクチュエータの待ち時間，警告ランプの点滅，リレー接点のチャタリング消去，アナログ変化量のサンプリング等々タイマが必要になるケースは数えきれません．

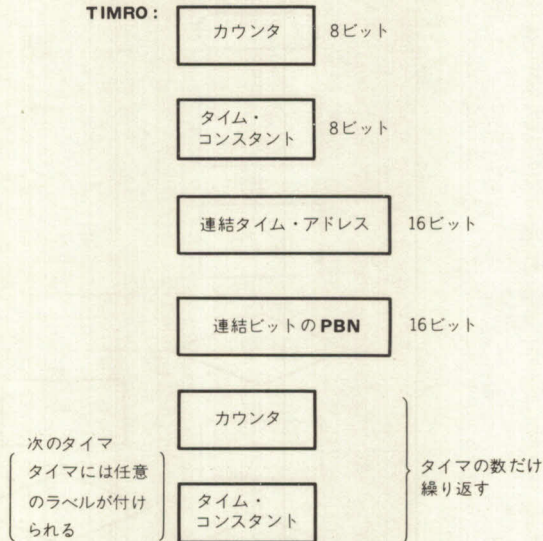
そこで，通常はハードウェアでタイマを作ったり，タイマ用 LSI を使用します．タイマ

〈図5.3〉 ソフトウェア・タイマのフローチャート・ブロック図

の数が少なければこれで対応できますが，10個以上の，それもソフト上で時間変更ができるタイマが必要となると大変になります．ハードウェアのタイマを1個にして，ソフトウェアでカウントして種々の時間に対応する方法も考えられますが，これは通常多くのタイマの並行動作ができません．

数10個のソフトウェアで変更可能なタイマが，全く独立の時間関係で並行動作できれば…便利ですね．ついでにタイマのカウントアップで，別のタイマを自動スタートしたり，I/OのビットをON/OFFしたりできたら……．この考え方をハードウェア風のブロック図にしたのが図5.3です．実際にはこのブロックはソフト的にメモリの中に作られます．そのメモリ・マップは図5.4のようにします．タイマ1個あたり6バイトで構成されます．

このダウン・カウンタをカウントするのは，ただ1個のハードウェア・タイマによるインタラプトです．ハードウェア・タイマによってソフトウェア・タイマの基準時間単位(1カウント当たりの時間)が決定されます．このインタラプトも周辺LSI(CTCや8253)を使用すればソフトでコントロールできます．

タイマの処理は，基本的にはタイマ・カウンタが0なら何もしない，0でなければダウン・カウントする，ダウン・カウントして0になったらカウント・アップと判断して連結されているタイマとI/Oのビットの処理をするという単純なものです．これをフローチャート

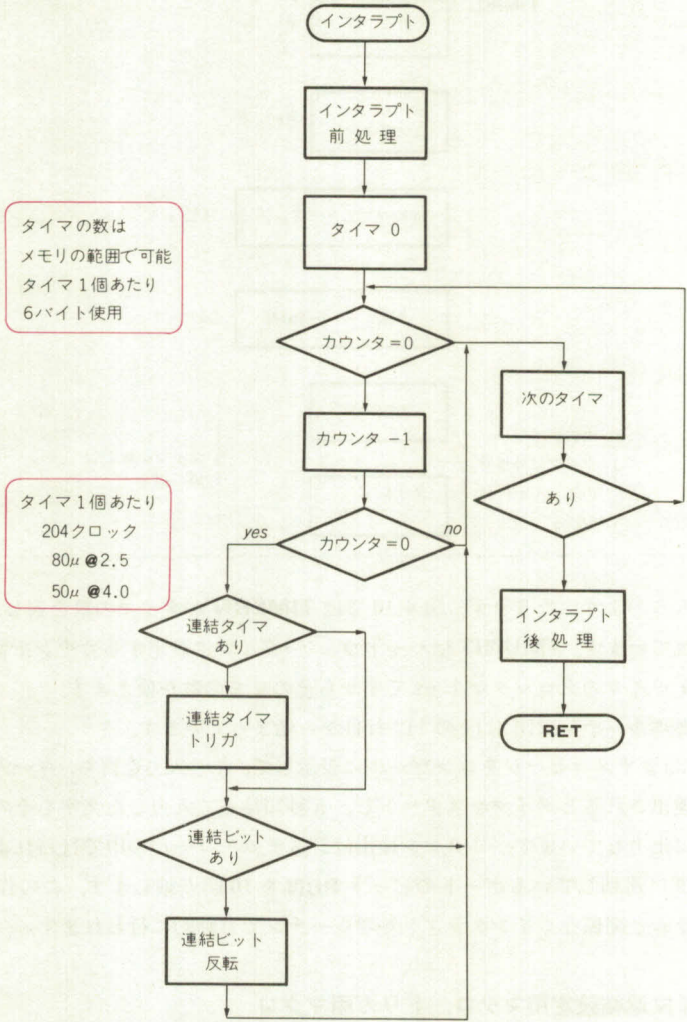
〈図 5.4〉ソフトウェア・タイマの作業エリア

で示せば図 5.5 のようになります．図 4.10 では **TIMERN** がタイマの数を表していて，**2** に **EQU** されています．**TMUNIT** はハードウェア・タイマに設定するカウント値です．ハードウェア・タイマのクロックが 1 μs ですからそのままの数が使えます．

タイマの処理ルーチンは図 4.11 の 118 行目から始まっています．

図 4.10 ではまずメッセージをコンソールに表示して，キー入力を待ち，キー入力されてリターンが検出されるとタイマをスタートし，5 秒間待って入力した文字をそのまま 2 回コンソールに出力しています．リターン検出はシステム・コールの中で行われます．タイマ 0 はその間に連動しているポートのビット **PBN** を 10 回点滅します．この作業はメイン・プログラムと関係なくインタラプト処理ルーチンで自動的に行われます．

5.2.1 タイマ連結設定用マクロ，トリガ用マクロ

図 4.10 の 19 行，21 行にタイマ連結用マクロ **CONJT** が呼び出されています．このマクロ定義は図 4.11 の 97 行からですが，これは単にタイマの作業用メモリに，パラメータの値を書き込むだけです．19 行の **CONJT** は，タイマ 0 に **PBN**(6 行で **EQU** されている) とタイマ 0 を連結し，時定数を 10 に設定せよという意味を表します．また，21 行の方はタ

〈図 5.5〉 ソフトウェア・タイマ処理のフローチャート

イマ **ECHOTM** に連結されている I/O とタイマをはずし，時定数を 100 に設定せよという意味です．

18 行のイニシャル・ルーチンで，時間単位が 50 ms にハード上でセットされていますか

ら，時定数はタイマ0が0.5秒，**ECHOTM**が5秒になります．タイマ0にはタイマ0自身が連結されているので，トリガ後カウント・アップするたびに**PBN**のビットをトグル（ひっくりかえす）してこのビットに接続されているランプをON/OFFします．**PBN**は1～255のポートに対して適用できます．連結タイマは0ページ以外のメモリ全域です．ただし，各タイマは連続した6バイトごとの配列になっていなければなりません．

時定数と連結指示は，省略すると前回のままとなります．1度は必ず指定しなければなりません．

トリガ用のマクロはタイマ名も省略できます．20行の**TRIGT**は省略形でタイマ0を指すようになっています．34行の場合は**ECHOTM**をトリガしています．トリガ時には，第2パラメータとしてタイマのカウンタに設定する値（0～255）が書けます．この値は直接カウンタに書かれ，時定数は変化しません．これを指定するのは，臨時に時間長を変える時と，途中でリセットする時(値を0にする)です．途中でリセットされたタイマはカウント・アップにならず，連結I/O，連結タイマともトリガされません．

タイマはカウント中にもトリガできます．このときは再びフルカウントからやりなおしになります．つまり，リトリガ機能になります．タイマはメモリですから，当然何時でも読み取れます．

この形式のソフト・タイマはプログラムの面倒な時間管理の部分を極めて簡単に書けるようにしてくれます．しかも，50 ms単位のタイマがメイン・プログラムの半分の時間でインタラプト処理をすると仮定した時，最悪のケース(すべてのタイマに連動するI/Oとタイマがあり，しかも全く同時にタイムアップとなった時)でも500チャネルの処理ができます．このためにRAM 3 Kバイトが必要ですが，もしハードウェアで500チャネルのどのI/Oにも連結できるタイマを作ったら相当大がかりなものになるでしょう．

5.3　同一マクロ呼び出しに対する5種の展開法

1.1節ではメイン・プログラムに手を入れなくても，マクロ定義のみを変更してスピード優先のコードやメモリ節約のコードが作り出せると書きました．ここではその具体的な例として4バイトの演算とビットのセットについて5種類の展開例を掲げ，各々のマクロ・パラメータと実行時のデータの扱いについて解説します．

スピード優先の方法では，できる限り生成コードの変化部分を多くしてマクロ展開時に解決してしまい，実行時の変化要素を減らします．逆にメモリ節約形ではマクロ・パラメ

ータによる生成コード変化をできる限り少なくし，処理の多くを実行ルーチンに負担させます．極端にメモリを節約する時には，**CALL** 命令のCD 1バイトをも削ってアドレスだけを生成しておき，実行時にそれを解釈するプログラムがそのアドレスを **CALL** する方法もあります．これは中間コード・インタプリタと呼ばれる方法ですが，ここまで含めてもメイン・プログラムの変更はほとんど不要になるマクロ定義が可能です．

各ソース・プログラム中には既出のマクロを使用しています．これらについては説明を省きます．

5.3.1 直接データを渡す方式

マクロを使うと，最も手軽にできるのがこの方式です．4バイトの加算には8バイトのデータを渡して，4バイトのデータを返してもらうわけですが，IX を使って8バイトのデータを渡すことはできます．ビット・セットの方は，すでに何回か出てきている毎回全部を展開する方式です．

〈図5.6〉各方式に共通のマクロ定義

```
 1:              .Z80
 2:     ; ｷｮｰﾂｰ ﾏｸﾛ
 3:     ;
 4:     ; ﾚｼﾞｽﾀ ﾚﾝｿﾞｸ ﾛｰﾄﾞ
 5:    LDIRR     MACRO     REGS,REG
 6:              IRPC      Q,REGS
 7:              LD        Q,(HL)
 8:              INC       HL
 9:              ENDM
10:              IFNB      <REG>
11:              LD        REG,(HL)  ;;ｲﾝｸﾘ ｼﾅｲ
12:              ENDIF
13:              ENDM
14:     ;
15:     ; ﾚﾝｿﾞｸ ｽﾄｱ
16:    STIR      MACRO     REGS,REG
17:              IRPC      Q,REGS
18:              LD        (HL),Q
19:              INC       HL
20:              ENDM
21:              IFNB      <REG>
22:              LD        (HL),REG
23:              ENDIF
24:              ENDM
25:     ;
```

4バイトの加算は実際には簡単なプログラムですが，これは乗除算でも同じようにできるために，サブルーチンとしておきます．

図5.6には共通に使うマクロを示します．5種類のすべての展開法に共通です．図5.7がメイン・プログラムです．

マクロ定義は図5.8に示す通りです．**HLIX**と**BCDE**で2つの4バイト・データを渡し，**HLIX**に答えが返ってきます．2度目のマクロ展開で25バイトとかなり大きなエリアを使います．

ビット・セットの方は直接展開で，ひとつのビット当たり6バイトです．

5.3.2 レジスタでアドレスを渡す方式

データを直接渡す方法は，データ長が6バイトや8バイトまたは可変長になると実際には使えないケースが出てきます．また，4バイトでも渡すデータとしては多すぎます．そこでデータのアドレスを渡す方式を考えます．これならデータ長に関係なく，常に2バイトでひとつのデータを表せます．可変長も扱えます．

アドレスを渡す方式にするとデータを受け取る作業は不要です．というのは，結果を入

〈図5.7〉マクロ ADDQ, ON のメイン・プログラム

```
 1:  ; ﾏｸﾛ !ADDQ,ON! ﾃｽﾄ
 2:  ; ﾁｮｸｾﾂ ﾃﾝｶｲ ｽﾙ
 3:  ;
 4:              .XLIST
 5:              INCLUDE B:G1.MAC ← 図5.6
 6:              INCLUDE B:G2.MAC ← 図5.8(a)
 7:              .LIST
 8:              ASEG
 9:              ORG       1000H
10:   QUAD       <AAA,BBB,CCC,DDD,EEE>
11:   QUAD       <JJJ,KKK>
12:   PBNDEF     5,<,,MON,TUE,WED>
13:   PBNDEF     8,<,RED,GRN,,BLU,BLK>
14:  ;
15:              CSEG
16:  START:      ADDQ      DDD,AAA,CCC
17:              ADDQ      BBB,JJJ,KKK
18:              ON        <RED,WED>
19:              ON        <TUE,BLU,MON>
20:              JP        START
21:  ;
22:              END       START
```

〈図5.8〉直接データを渡すマクロ

(a) マクロ定義のソース・リスト

```
  1:  ; ﾁｮｸｾﾂ ﾃﾞｰﾀ ｦ ﾜﾀｽ
  2:  ;
  3:  CHECK    MACRO   MNAME,EM
  4:           IF      20H AND NOT TYPE MNAME
  5:  MNAME    EQU     $+3
  6:           ENDIF
  7:           IF      MNAME EQ $+3
  8:           JP      EM
  9:           ENDM
 10:  ;
 11:  ADDQ     MACRO   S1,S2,DST
 12:           LOCAL   EM
 13:           LD      IX,(S1)
 14:           LD      HL,(S1+2)
 15:           LD      DE,(S2)
 16:           LD      BC,(S2+2)
 17:           CALL    ADDQ
 18:           LD      (DST),IX
 19:           LD      (DST+2),HL
 20:           CHECK   ADDQ,EM
 21:           ADD     IX,DE
 22:           ADC     HL,BC
 23:           RET
 24:  EM:
 25:           ENDIF
 26:           ENDM
 27:  ;
 28:  ON       MACRO   PBNS        ;ｽﾍﾞﾃ ﾏｲｶｲ ﾃﾝｶｲ ｻﾚﾙ
 29:           IRP     QQ,<PBNS>
 30:           IN      A,(HIGH QQ)
 31:           SET     LOW QQ,A
 32:           OUT     (HIGH QQ),A
 33:           ENDM
 34:           ENDM
 35:  ;
 36:  PBNDEF   MACRO   PORT,BNAME
 37:  TEMP     DEFL    0
 38:           IRP     Q,<BNAME>
 39:           IFNB    <Q>
 40:  Q        EQU     PORT*256+TEMP
 41:           ENDIF
 42:  TEMP     DEFL    TEMP+1
 43:           ENDM
 44:           ENDM
 45:  ;
 46:  QUAD     MACRO   NAMES
 47:           IRP     QQ,<NAMES>
 48:  QQ:      DS      4
 49:           ENDM
 50:           ENDM
 51:  ;
```

(b) アセンブル・リスト

```
                                ; マクロ ADDQ,ON テスト
                                ; チョクセツ テンカイ スル
                                ;
                                        .LIST
0000'                                   ASEG
                                        ORG     1000H
                                 QUAD   <AAA,BBB,CCC,DDD,EEE>
1000                      +     AAA:    DS      4
1004                      +     BBB:    DS      4
1008                      +     CCC:    DS      4
100C                      +     DDD:    DS      4
1010                      +     EEE:    DS      4
                                 QUAD   <JJJ,KKK>
1014                      +     JJJ:    DS      4
1018                      +     KKK:    DS      4
                                 PBNDEF 5,<,,MON,TUE,WED>
                                 PBNDEF 8,<,RED,GRN,,BLU,BLK>
                                ;
101C                                    CSEG
0000'                           START:  ADDQ    DDD,AAA,CCC
0000'   DD 2A 100C        +             LD      IX,(DDD)
0004'   2A 100E           +             LD      HL,(DDD+2)
0007'   ED 5B 1000        +             LD      DE,(AAA)
000B'   ED 4B 1002        +             LD      BC,(AAA+2)
000F'   CD 001C'          +             CALL    ADDQ
0012'   DD 22 1008        +             LD      (CCC),IX
0016'   22 100A           +             LD      (CCC+2),HL
0019'   C3 0021           +             JP      ..0000
001C'   DD 19             +             ADD     IX,DE
001E'   ED 4A             +             ADC     HL,BC
0020'   C9                +             RET
0021'                     +     ..0000:
                                        ADDQ    BBB,JJJ,KKK
0021'   DD 2A 1004        +             LD      IX,(BBB)
0025'   2A 1006           +             LD      HL,(BBB+2)
0028'   ED 5B 1014        +             LD      DE,(JJJ)
002C'   ED 4B 1016        +             LD      BC,(JJJ+2)
0030'   CD 001C'          +             CALL    ADDQ
0033'   DD 22 1018        +             LD      (KKK),IX
0037'   22 101A           +             LD      (KKK+2),HL
                                        ON      <RED,WED>
003A'   DB 08             +             IN      A,(HIGH RED)
003C'   CB CF             +             SET     LOW RED,A
003E'   D3 08             +             OUT     (HIGH RED),A
0040'   DB 05             +             IN      A,(HIGH WED)
0042'   CB E7             +             SET     LOW WED,A
0044'   D3 05             +             OUT     (HIGH WED),A
                                        ON      <TUE,BLU,MON>
```

```
0046'    DB 05          +          IN      A,(HIGH TUE)
0048'    CB DF          +          SET     LOW TUE,A
004A'    D3 05          +          OUT     (HIGH TUE),A
004C'    DB 08          +          IN      A,(HIGH BLU)
004E'    CB E7          +          SET     LOW BLU,A
0050'    D3 08          +          OUT     (HIGH BLU),A
0052'    DB 05          +          IN      A,(HIGH MON)
0054'    CB D7          +          SET     LOW MON,A
0056'    D3 05          +          OUT     (HIGH MON),A
0058'    C3 0000'                  JP      START
                           ;
                                   END     START
```

れるべきアドレスをもサブルーチンの方へ渡してしまえるからです．結果をメモリに戻す作業はサブルーチン内で済ませます．

ビット･セットの方は **PBN** をレジスタに置いて渡すようにしました．この方式では展開するたびにビットあたり6バイトでプログラムは処理ルーチンの分だけ増えた計算になります．しかし，もし処理内容が複雑なものであれば毎回展開するよりサブルーチン化した方がプログラムが小さくなります．ここでの例は，アドレスでなくても(**PBN** のような含みのある数でも)直接展開でなくサブルーチン処理する方法もとれるという意味を持っています．

この方式のマクロ定義を図5.9(a)に，展開リストを同図(b)に示します．**ADDQ** の方は展開ごとに14バイトになり，データ渡しよりもかなり減りました．マクロ呼び出しをするメイン・プログラムは図5.7と全く同じでインクルードするファイルだけが変更になります．

5.3.3 インライン・パラメータでアドレスを渡す方式

これまでの方式では，レジスタを介してサブルーチンに情報を渡していました．しかし，レジスタには量的に限りがあって，パラメータの数が多いときは，たとえアドレスといえども渡しきれなくなります．また，大量のレジスタを情報伝達に使用するとメイン・プログラム側で **PUSH/POP** の回数が増え，トータルのメモリ量が多く費やされます．

そこで，今度はレジスタを全く介せず，**CALL** 命令の後に続くデータ，いわゆるインライン･パラメータでアドレスを渡すようにします．こうすれば，レジスタの **PUSH/POP** をメイン・プログラム側でやる必要はなく，すべてサブルーチン側にゲタを預けられます．これまでの過程でサブルーチンはどんどん大きくなってきますが，サブルーチンは一度だけしか展開されませんから，サブルーチンが大きくなることは問題になりません．

〈図5.9〉 アドレスをレジスタで渡すマクロ

(a) ソース・リスト

```
  1:   ; ｱﾄﾞﾚｽ ｦ ﾚｼﾞｽﾀ ﾃﾞ ﾜﾀｽ
  2:   ;
  3:   CHECK     MACRO    MNAME,EM
  4:             IF       20H AND NOT TYPE MNAME
  5:   MNAME     EQU      $+3
  6:             ENDIF
  7:             IF       MNAME EQ $+3
  8:             JP       EM
  9:             ENDM
 10:   ;
 11:   ADDQ      MACRO    S1,S2,DST
 12:             LOCAL    EM
 13:             LD       IX,S1     ;;|3|ﾂ ﾉ ｱﾄﾞﾚｽ ｦ ﾜﾀｼﾃ
 14:             LD       HL,S2
 15:             LD       IY,DST
 16:             CALL     ADDQ      ;;ｼｮﾘﾙｰﾁﾝ ﾍ ﾄﾌﾞ
 17:             CHECK    ADDQ,EM
 18:             LDIRR    EDC,B
 19:             PUSH     BC
 20:             PUSH     DE
 21:             EX       (SP),IX
 22:             POP      HL
 23:             LDIRR    EDC,B
 24:             POP      HL
 25:             ADD      IX,DE
 26:             ADC      HL,BC
 27:             LD       C,L
 28:             LD       B,H
 29:             PUSH     IX
 30:             POP      DE
 31:             PUSH     IY
 32:             POP      HL
 33:             STIR     EDC,B
 34:             RET
 35:   EM:
 36:             ENDIF
 37:             ENDM
 38:   ;
 39:   ON        MACRO    PBNS
 40:             LOCAL    EM
 41:             IRP      QQ,<PBNS>
 42:             LD       HL,QQ     ;;PBNｦ HL ﾆ ｵｲﾃ
 43:             CALL     ONBIT     ;;ｼｮﾘﾙｰﾁﾝ ﾍ
 44:             ENDM
 45:             CHECK    ONBIT,EM
 46:             INC      B         ;ﾋﾞｯﾄ ｲﾁ ｦ ｹｲｻﾝ
 47:             LD       A,80H
```

```
48:             RLCA
49:             DJNZ     $-1
50:             IN       E,(C)
51:             OR       E            ;ｾｯﾄ ｽﾙ
52:             OUT      (C),A
53:             RET
54:     EM:
55:             ENDIF
56:             ENDM
57:     ;
58:     PBNDEF  MACRO    PORT,BNAME
59:     TEMP    DEFL     0
60:             IRP      Q,<BNAME>
61:             IFNB     <Q>
62:     Q       EQU      TEMP*256+PORT ;;HﾄL ﾉ ﾊﾞｲﾄ ﾁｭｰｲ
63:             ENDIF
64:     TEMP    DEFL     TEMP+1
65:             ENDM
66:             ENDM
67:     ;
68:     QUAD    MACRO    NAMES
69:             IRP      QQ,<NAMES>
70:     QQ:     DS       4
71:             ENDM
72:             ENDM
73:     ;
```

(b) アセンブル・リスト

```
                                ; ﾏｸﾛ ADDQ,ON ﾃｽﾄ
                                ; ﾚｼﾞｽﾀ ﾃﾞ ｱﾄﾞﾚｽ ｦ ﾜﾀｽ
                                ;
                                        .LIST
 0000'                                  ASEG
                                        ORG     1000H
                                 QUAD   <AAA,BBB,CCC,DDD,EEE>
 1000                      +    AAA:    DS      4
 1004                      +    BBB:    DS      4
 1008                      +    CCC:    DS      4
 100C                      +    DDD:    DS      4
 1010                      +    EEE:    DS      4
                                 QUAD   <JJJ,KKK>
 1014                      +    JJJ:    DS      4
 1018                      +    KKK:    DS      4
                                 PBNDEF 5,<,,MON,TUE,WED>
                                 PBNDEF 8,<,RED,GRN,,BLU,BLK>
                                ;
 101C                                   CSEG
 0000'                          START:  ADDQ    DDD,AAA,CCC
 0000'   DD 21 100C        +            LD      IX,DDD
 0004'   21 1000           +            LD      HL,AAA
 0007'   FD 21 1008        +            LD      IY,CCC
 000B'   CD 0011'          +            CALL    ADDQ
 000E'   C3 0039'          +            JP      ..0000
 0011'   5E                +            LD      E,(HL)
 0012'   23                +            INC     HL
 0013'   56                +            LD      D,(HL)
```

```
0014'  23              +          INC     HL
0015'  4E              +          LD      C,(HL)
0016'  23              +          INC     HL
0017'  46              +          LD      B,(HL)
0018'  C5              +          PUSH    BC
0019'  D5              +          PUSH    DE
001A'  DD E3           +          EX      (SP),IX
001C'  E1              +          POP     HL
001D'  5E              +          LD      E,(HL)
001E'  23              +          INC     HL
001F'  56              +          LD      D,(HL)
0020'  23              +          INC     HL
0021'  4E              +          LD      C,(HL)
0022'  23              +          INC     HL
0023'  46              +          LD      B,(HL)
0024'  E1              +          POP     HL
0025'  DD 19           +          ADD     IX,DE
0027'  ED 4A           +          ADC     HL,BC
0029'  4D              +          LD      C,L
002A'  44              +          LD      B,H
002B'  DD E5           +          PUSH    IX
002D'  D1              +          POP     DE
002E'  FD E5           +          PUSH    IY
0030'  E1              +          POP     HL
0031'  73              +          LD      (HL),E
0032'  23              +          INC     HL
0033'  72              +          LD      (HL),D
0034'  23              +          INC     HL
0035'  71              +          LD      (HL),C
0036'  23              +          INC     HL
0037'  70              +          LD      (HL),B
0038'  C9              +          RET
0039'                  +    ..0000:
                                  ADDQ    BBB,JJJ,KKK
0039'  DD 21 1004      +          LD      IX,BBB
003D'  21 1014         +          LD      HL,JJJ
0040'  FD 21 1018      +          LD      IY,KKK
0044'  CD 0011'        +          CALL    ADDQ
                                  ON      <RED,WED>
0047'  21 0108         +          LD      HL,RED
004A'  CD 0056'        +          CALL    ONBIT
004D'  21 0405         +          LD      HL,WED
0050'  CD 0056'        +          CALL    ONBIT
0053'  C3 0062'        +          JP      ..0002
0056'  04              +          INC     B          ;ビット イチ ヲ ケイサン
0057'  3E 80           +          LD      A,80H
0059'  07              +          RLCA
005A'  10 FD           +          DJNZ    $-1
005C'  ED 58           +          IN      E,(C)
005E'  B3              +          OR      E          ;セット スル
005F'  ED 79           +          OUT     (C),A
0061'  C9              +          RET
0062'                  +    ..0002:
                                  ON      <TUE,BLU,MON>
0062'  21 0305         +          LD      HL,TUE
0065'  CD 0056'        +          CALL    ONBIT
0068'  21 0408         +          LD      HL,BLU
006B'  CD 0056'        +          CALL    ONBIT
006E'  21 0205         +          LD      HL,MON
0071'  CD 0056'        +          CALL    ONBIT
0074'  C3 0000'                   JP      START
                            ;
                                  END     START
```

ビット・セットの方は，前項ではビットの数だけ **CALL** を繰り返しています．これは，パラメータが何個渡されるか決まっていない (**IRP** で展開しているので何10個かは可能) のでレジスタ経由でパラメータを渡す以上この手しかありません．インライン・パラメータではパラメータの数は制限されませんから，パラメータの終わりが判断できるようにさえなっていれば困る要素はありません．この方式でのマクロ定義と展開結果を図5.10に示します．メイン・プログラムは前項と同じです．

この方式では **ADDQ** が9バイト，**ON** が1ビットのとき7バイト，3ビットのとき11バイトで，**ON** についてはビットが少ない時は不利です．パラメータの終わりを示すのに **8000H** を付加したのでこのようになりました．効率を上げるには，最後のパラメータに **8000H** を **OR** した形でマクロ定義をすれば2バイト節約できます．このためには，**IRP** の中で最後のパラメータを展開しているという判断をしなければなりません．

または，1ビットのみのときは5.3.1項の方式に展開してしまってもよいでしょう．

図5.10の例では，**ADDQ** も **ON** も **CALL** のあとに **DW** が続いています．**ADDQ** ではデータのアドレス，**ON** では **PBN** が入っているわけですが，両方とも2バイトの情報なのでこの形になりました．しかし，パラメータがバイト単位での情報であれば **DB** になっても全く支障ありません．ただし，**CALL** されたサブルーチンの側ではサブルーチンからのリターン・アドレスを間違えないようにしないとプログラムが暴走してしまいます．

もしマクロ命令のみを使用したとすれば，**CALL** 以外はすべてデータの列になるというのがこの方式の特徴です．また，逆アセンブルのような方法では解析しにくくなる性質もあります．それでも，すべてマクロ命令で書くわけにはいきませんから，全く解読不可能というわけではありません．**CALL** 命令のコード **CD** もデータと一致する可能性があるとはいえ，かなりたよりになります．

5.3.4　アドレス・リストによるインタプリタ

前項までは呼び出し側のメイン・プログラムは図5.7と同じでした．サブルーチンとのデータのやり取りが変わったこと以外は本質的に同じと考えてよいものです．この項ではそれが大きく変わって機械語コードを使用しないで目的のサブルーチンを呼び出すいわゆるインタプリタ方式に相当するものになります．

インタプリタですから，命令の列 (**ADDQ** や **ON**) の中に通常の機械語命令を混用することは不可能です．とはいっても，必要なすべての機能をインタプリタで解釈するのは大変すぎますから，機械語命令起動用の命令をインタプリタのコードで定義しておきます．

〈図5.10〉インライン・パラメータでアドレスを渡すマクロ

(a) ソース・リスト

```
 1:  ; ｱﾄﾞﾚｽ ｦ |CALL|ﾆﾂﾂﾞｸ ｲﾝﾗｲﾝ
 2:  ; ﾊﾟﾗﾒｰﾀ ﾄｼﾃ ﾜﾀｽ
 3:  ;
 4:  CHECK     MACRO     MNAME,EM
 5:            IF        20H AND NOT TYPE MNAME
 6:  MNAME     EQU       $+3
 7:            ENDIF
 8:            IF        MNAME EQ $+3
 9:            JP        EM
10:            ENDM
11:  ;
12:  ADDQ      MACRO     S1,S2,DST
13:            LOCAL     EM
14:            CALL      ADDQ
15:            DW        S1,S2,DST ;;ｲﾝﾗｲﾝ ﾊﾟﾗﾒﾀ
16:            CHECK     ADDQ,EM
17:            EX        (SP),HL  ;ﾊﾟﾗﾒﾀ ﾎﾟｲﾝﾀ
18:            PUSH      DE
19:            PUSH      BC        ;ｾｰﾌﾞ
20:            LDIRR     EDCB      ;ﾃﾞｰﾀ ﾎﾟｲﾝﾀ
21:            PUSH      HL        ;|P| ﾎﾟｲﾝﾀ ｾｰﾌﾞ
22:            EX        DE,HL
23:            PUSH      BC
24:            POP       IX
25:            LDIRR     EDC,B
26:            PUSH      BC
27:            PUSH      DE
28:            EX        (SP),IX
29:            POP       HL
30:            LDIRR     EDC,B
31:            POP       HL
32:            ADD       IX,DE     ;
33:            ADC       HL,BC
34:            LD        C,L
35:            LD        B,H
36:            POP       HL
37:            LDIRR     ED
38:            PUSH      HL        ;ﾘﾀﾝ ｱﾄﾞﾚｽ
39:            EX        DE,HL     ;ｺﾀｴ ﾎﾟｲﾝﾀ
40:            PUSH      IX
41:            POP       DE
42:            STIR      EDC,B
43:            POP       HL
44:            POP       BC
45:            POP       DE
46:            EX        (SP),HL
47:            RET
```

```
48:  EM:
49:             ENDIF
50:             ENDM
51:  ;
52:  ON         MACRO    PBNS
53:             LOCAL    EM
54:             CALL     ONBIT
55:             DW       PBNS,8000H
56:             CHECK    ONBIT,EM
57:             EX       (SP),HL  ;|P|ﾎﾟｲﾝﾀ
58:             PUSH     DE
59:             PUSH     BC
60:             PUSH     AF
61:             LDIRR    CB
62:             BIT      7,B
63:             JR       NZ,EM-5
64:             INC      B
65:             LD       A,80H
66:             RLCA
67:             DJNZ     $-1
68:             IN       E,(C)
69:             OR       E
70:             OUT      (C),A
71:             JR       ONBIT+4
72:             POP      AF
73:             POP      BC
74:             POP      DE
75:             EX       (SP),HL
76:             RET
77:  EM:
78:             ENDIF
79:             ENDM
80:  ;
81:  PBNDEF     MACRO    PORT,BNAME
82:  TEMP       DEFL     0
83:             IRP      Q,<BNAME>
84:             IFNB     <Q>
85:  Q          EQU      TEMP*256+PORT
86:             ENDIF
87:  TEMP       DEFL     TEMP+1
88:             ENDM
89:             ENDM
90:  ;
91:  QUAD       MACRO    NAMES
92:             IRP      QQ,<NAMES>
93:  QQ:        DS       4
94:             ENDM
95:             ENDM
96:  ;
```

(b) アセンブル・リスト

```
                          ; ﾏｸﾛ ADDQ,ON ﾃｽﾄ
                          ; ｲﾝﾗｲﾝ ﾊﾟﾗﾒｰﾀ ﾜﾀｼ
                          ;
                                  .LIST
0000'                             ASEG
                                  ORG     1000H
                           QUAD   <AAA,BBB,CCC,DDD,EEE>
1000                  +   AAA:    DS      4
1004                  +   BBB:    DS      4
1008                  +   CCC:    DS      4
100C                  +   DDD:    DS      4
1010                  +   EEE:    DS      4
                           QUAD   <JJJ,KKK>
1014                  +   JJJ:    DS      4
1018                  +   KKK:    DS      4
                           PBNDEF 5,<,,MON,TUE,WED>
                           PBNDEF 8,<,RED,GRN,,BLU,BLK>
                          ;
101C                              CSEG
0000'                     START:  ADDQ        DDD,AAA,CCC
0000'   CD 000C'          +       CALL        ADDQ
0003'   100C 1000         +       DW          DDD,AAA,CCC
0007'   1008              +
0009'   C3 004C'          +       JP          ..0000
000C'   E3                +       EX          (SP),HL  ;ﾊﾟﾗﾒﾀ ﾎﾟｲﾝﾀ
000D'   D5                +       PUSH        DE
000E'   C5                +       PUSH        BC       ;ｾｰﾌﾞ
000F'   5E                +       LD          E,(HL)
0010'   23                +       INC         HL
0011'   56                +       LD          D,(HL)
0012'   23                +       INC         HL
0013'   4E                +       LD          C,(HL)
0014'   23                +       INC         HL
0015'   46                +       LD          B,(HL)
0016'   23                +       INC         HL
0017'   E5                +       PUSH        HL       ;P ﾎﾟｲﾝﾀ ｾｰﾌﾞ
0018'   EB                +       EX          DE,HL
0019'   C5                +       PUSH        BC
001A'   DD E1             +       POP         IX
001C'   5E                +       LD          E,(HL)
001D'   23                +       INC         HL
001E'   56                +       LD          D,(HL)
001F'   23                +       INC         HL
0020'   4E                +       LD          C,(HL)
0021'   23                +       INC         HL
0022'   46                +       LD          B,(HL)
0023'   C5                +       PUSH        BC
0024'   D5                +       PUSH        DE
0025'   DD E3             +       EX          (SP),IX
0027'   E1                +       POP         HL
0028'   5E                +       LD          E,(HL)
0029'   23                +       INC         HL
002A'   56                +       LD          D,(HL)
002B'   23                +       INC         HL
002C'   4E                +       LD          C,(HL)
```

```
002D'   23          +           INC     HL
002E'   46          +           LD      B,(HL)
002F'   E1          +           POP     HL
0030'   DD 19       +           ADD     IX,DE     ;
0032'   ED 4A       +           ADC     HL,BC
0034'   4D          +           LD      C,L
0035'   44          +           LD      B,H
0036'   E1          +           POP     HL
0037'   5E          +           LD      E,(HL)
0038'   23          +           INC     HL
0039'   56          +           LD      D,(HL)
003A'   23          +           INC     HL
003B'   E5          +           PUSH    HL        ;ﾘﾀﾝ ｱﾄﾞﾚｽ
003C'   EB          +           EX      DE,HL     ;ｺﾀｴ ﾎﾟｲﾝﾀ
003D'   DD E5       +           PUSH    IX
003F'   D1          +           POP     DE
0040'   73          +           LD      (HL),E
0041'   23          +           INC     HL
0042'   72          +           LD      (HL),D
0043'   23          +           INC     HL
0044'   71          +           LD      (HL),C
0045'   23          +           INC     HL
0046'   70          +           LD      (HL),B
0047'   E1          +           POP     HL
0048'   C1          +           POP     BC
0049'   D1          +           POP     DE
004A'   E3          +           EX      (SP),HL
004B'   C9          +           RET
004C'               +   ..0000:
                                ADDQ    BBB,JJJ,KKK
004C'   CD 000C'    +           CALL    ADDQ
004F'   1004 1014   +           DW      BBB,JJJ,KKK
0053'   1018        +
                                ON      <RED,WED>
0055'   CD 0061'    +           CALL    ONBIT
0058'   0108 0405   +           DW      RED,WED,8000H
005C'   8000        +
005E'   C3 007F'    +           JP      ..0002
0061'   E3          +           EX      (SP),HL ;Pﾎﾟｲﾝﾀ
0062'   D5          +           PUSH    DE
0063'   C5          +           PUSH    BC
0064'   F5          +           PUSH    AF
0065'   4E          +           LD      C,(HL)
0066'   23          +           INC     HL
0067'   46          +           LD      B,(HL)
0068'   23          +           INC     HL
0069'   CB 78       +           BIT     7,B
006B'   20 0D       +           JR      NZ,..0002-5
006D'   04          +           INC     B
006E'   3E 80       +           LD      A,80H
0070'   07          +           RLCA
0071'   10 FD       +           DJNZ    $-1
0073'   ED 58       +           IN      E,(C)
0075'   B3          +           OR      E
0076'   ED 79       +           OUT     (C),A
0078'   18 EB       +           JR      ONBIT+4
007A'   F1          +           POP     AF
```

```
007B'   C1              +               POP     BC
007C'   D1              +               POP     DE
007D'   E3              +               EX      (SP),HL
007E'   C9              +               RET
007F'                   +       ..0002:
                                        ON      <TUE,BLU,MON>
007F'   CD 0061'        +               CALL    ONBIT
0082'   0305 0408       +               DW      TUE,BLU,MON,8000H
0086'   0205 8000       +
008A'   C3 0000'                        JP      START
                                ;
                                        END     START
```

また，ジャンプ命令も機械語では実行できませんから，インタプリタのコードにしておきます．

上記の2種類のインタプリタの命令(インタコードと呼ぶ)は不可欠なので，これらを追加したマクロ定義とその展開結果を図5.12に，また追加命令も含めてテストするためのメイン・プログラムを図5.11に示します．

マクロ定義の中で，**QUAD** と **PBNDEF** は全く変わっていません．**ADDQ** と **ON** も最初と最後が少し変わっただけで内容として大きな変化はありません．このインタプリタではDEをPCとして動作していますから，サブルーチン内では5.3.3項の方式よりも簡単にデータのアドレスを取り出せます．その分サブルーチンが小さくなっています．

インタプリタの本体はマクロ **INTPRI** で起動され，インタコードに関してはこの部分が管理して指定されたアドレスのサブルーチンをコール〔実際は **JP　(HL)**〕し，サブルーチンの **RET** は再びこの部分に制御を返すようになります．

MACHIN はこのインタプリタから抜け出して，その次のアドレスから直接機械語として実行するためのインタコードで，その後の **CALL** まで機械語として実行されます．**CALL** で **TO_INTPRI** へ飛ぶと再びインタプリタが起動されて，その次のアドレスからインタコード解釈になります．この機械語テスト用の4行を除けば，メイン・プログラムは1行も増えていません．ただ，**CSEG** が **INTPRI** に変わっただけです．また，**JMP** はインタコードの中でのジャンプに変わっています．

要は前項の **CALL** とデータの組み合わせから **CALL** 命令の **CD** を取り除いた構成を，解釈用のプログラムを追加して実行可能にしたということです．当然，マクロ呼び出しごとに1バイトずつ少なくて済みます．

さて，このプログラムの実行ファイルかROMを見て動作が読み取れるでしょうか．マ

クロ命令を増やして機械語使用をできるだけ避ければ逆アセンブルによる解析は全く不可能でしょう.

〈図5.11〉 アドレス・リストによるインタプリタ

```
 1:  ; ﾏｸﾛ !ADDQ,ON,JMP,MACHIN! ﾃｽﾄ
 2:  ; ｱﾄﾞﾚｽ ﾘｽﾄ ﾆ ﾖﾙ ｲﾝﾀﾌﾟﾘﾀ
 3:  ;
 4:              .XLIST
 5:              INCLUDE B:G1.MAC
 6:              INCLUDE B:G5.MAC ← 図5.12(a)
 7:              .LIST
 8:              ASEG
 9:              ORG     1000H
10:      QUAD    <AAA,BBB,CCC,DDD,EEE>
11:      QUAD    <JJJ,KKK>
12:      PBNDEF  5,<,,MON,TUE,WED>
13:      PBNDEF  8,<,RED,GRN,,BLU,BLK>
14:  ;
15:              INTPRI                  ;ｲﾝﾀﾌﾟﾘﾀ ﾖﾋﾞﾀﾞｼ
16:              ADDQ    DDD,AAA,CCC
17:              ADDQ    BBB,JJJ,KKK
18:              MACHIN
19:              LD      A,5
20:              LD      (EEE),A
21:              CALL    TO_INTPRI   ;ｲﾝﾀﾌﾟﾘﾀﾍ
22:              ON      <RED,WED>
23:              ON      <TUE,BLU,MON>
24:              JMP     ICODE     ;!INTPRI!ﾆ ﾌｸﾏﾚﾃ ｲﾙ
25:  ;
26:              END     START
```

〈図5.12〉 インタコードを追加したマクロ定義

(a) ソース・リスト

```
 1:  ; !CALL!ﾗﾂｶﾜｽﾞ ｱﾄﾞﾚｽ ﾉﾐ ﾃﾞ
 2:  ; ｼｮﾘﾙｰﾁﾝ ﾍ ﾄﾌﾞ
 3:  ;
 4:  CHECK    MACRO   MNAME,EM
 5:           IF      20H AND NOT TYPE MNAME
 6:  MNAME    EQU     $+4
 7:           ENDIF
 8:           IF      MNAME EQ $+4
 9:           DW      JMP,EM   ;;ｲﾝﾀﾌﾟﾘﾀ ﾉ ｼﾞｬﾑﾌﾟ
10:           ENDM
11:  ;
12:  ADDQ     MACRO   S1,S2,DST
13:           LOCAL   EM
```

```
14:            DW      ADDQ        ;;ﾄﾋﾞｻｷ ｱﾄﾞﾚｽ
15:            DW      S1,S2,DST   ;;ｲﾝﾗｲﾝ ﾊﾟﾗﾒﾀ
16:            CHECK   ADDQ,EM
17:            EX      DE,HL
18:            LDIRR   EDCB        ;ﾃﾞｰﾀ ﾎﾟｲﾝﾀ
19:            PUSH    HL          ;|P| ﾎﾟｲﾝﾀ ｾｰﾌﾞ
20:            EX      DE,HL
21:            PUSH    BC
22:            POP     IX
23:            LDIRR   EDC,B
24:            PUSH    BC
25:            PUSH    DE
26:            EX      (SP),IX
27:            POP     HL
28:            LDIRR   EDC,B
29:            POP     HL
30:            ADD     IX,DE
31:            ADC     HL,BC
32:            LD      C,L
33:            LD      B,H
34:            POP     HL
35:            LDIRR   ED
36:            PUSH    HL          ;ﾘﾀﾝ ｱﾄﾞﾚｽ
37:            EX      DE,HL       ;ｺﾀｴ ﾎﾟｲﾝﾀ
38:            PUSH    IX
39:            POP     DE
40:            STIR    EDC,B
41:            POP     DE          ;ｲﾝﾀ ﾎﾟｲﾝﾀ
42:            RET
43:    EM:
44:            ENDIF
45:            ENDM
46:    ;
47:    ON      MACRO   PBNS
48:            LOCAL   EM
49:            DW      ONBIT,PBNS,8000H
50:            CHECK   ONBIT,EM
51:            EX      DE,HL
52:            LDIRR   CB
53:            BIT     7,B
54:            JR      NZ,EM-2
55:            INC     B
56:            LD      A,80H       ;|OFF|ﾉﾄｷ ﾊ |7FH
57:            RLCA
58:            DJNZ    $-1
59:            IN      E,(C)
60:            OR      E           ;|OFF| ﾉﾄｷ ﾊ |AND
61:            OUT     (C),A
62:            JR      ONBIT+1
63:            EX      DE,HL
```

```
 64:             RET
 65:    EM:
 66:             ENDIF
 67:             ENDM
 68:    ;
 69:    INTPRI   MACRO
 70:             CSEG
 71:    START:   LD         DE,ICODE    ;ｲﾝﾀｺｰﾄﾞ ﾊｼﾞﾒ
 72:             LD         BC,$ '
 73:             PUSH       BC
 74:             EX         DE,HL
 75:             LDIRR      ED
 76:             EX         DE,HL     ;|DE:pointer HL:jmp ads
 77:             JP         (HL)
 78:    TO_INTPRI: POP      DE        ;|pointer recover
 79:             JR         START+3
 80:    JMP:     EX         DE,HL
 81:             LDIRR      E,D
 82:             RET
 83:    MACHIN:  INC        SP
 84:             INC        SP
 85:             EX         DE,HL
 86:             JP         (HL)      ;ｲﾝﾀ|PC| ｦ ｼﾞｯｺｳ
 87:    ICODE:
 88:             ENDM
 89:    ;
 90:    JMP      MACRO      ADS
 91:             DW         JMP,ADS
 92:             ENDM
 93:    ;
 94:    MACHIN   MACRO                ;ｺｺﾖﾘ ｷｶｲｺﾞ ﾒｲﾚｲ
 95:             DW         MACHIN
 96:             ENDM
 97:    ;
 98:    PBNDEF   MACRO      PORT,BNAME
 99:    TEMP     DEFL       0
100:             IRP        Q,<BNAME>
101:             IFNB       <Q>
102:    Q        EQU        TEMP*256+PORT
103:             ENDIF
104:    TEMP     DEFL       TEMP+1
105:             ENDM
106:             ENDM
107:    ;
108:    QUAD     MACRO      NAMES
109:             IRP        QQ,<NAMES>
110:    QQ:      DS         4
111:             ENDM
112:             ENDM
113:    ;
```

(b) アセンブル・リスト

```
                                ; ﾏｸﾛ ADDQ,ON,JMP,MACHIN ﾃｽﾄ
                                ; ｱﾄﾞﾚｽ ﾘｽﾄ ﾆ ﾖﾙ ｲﾝﾀﾌﾟﾘﾀ
                                ;
                                        .LIST
0000'                                   ASEG
                                        ORG     1000H
                                 QUAD   <AAA,BBB,CCC,DDD,EEE>
1000                         +  AAA:    DS      4
1004                         +  BBB:    DS      4
1008                         +  CCC:    DS      4
100C                         +  DDD:    DS      4
1010                         +  EEE:    DS      4
                                 QUAD   <JJJ,KKK>
1014                         +  JJJ:    DS      4
1018                         +  KKK:    DS      4
                                 PBNDEF 5,<,,MON,TUE,WED>
                                 PBNDEF 8,<,RED,GRN,,BLU,BLK>
                                ;
                                        INTPRI                  ;ｲﾝﾀﾌﾟﾘﾀ ﾖﾋﾞﾀﾞｼ
101C                         +          CSEG
0000'   11 001A'             +  START:  LD      DE,ICODE        ;ｲﾝﾀｺｰﾄﾞ ﾊｼﾞﾒ
0003'   01 0003'             +          LD      BC,$
0006'   C5                   +          PUSH    BC
0007'   EB                   +          EX      DE,HL
0008'   5E                   +          LD      E,(HL)
0009'   23                   +          INC     HL
000A'   56                   +          LD      D,(HL)
000B'   23                   +          INC     HL
000C'   EB                   +          EX      DE,HL           ;DE:pointer HL:jmp ads
000D'   E9                   +          JP      (HL)
000E'   D1                   +  TO_INTPRI: POP  DE              ;pointer recover
000F'   18 F2                +          JR      START+3
0011'   EB                   +  JMP:    EX      DE,HL
0012'   5E                   +          LD      E,(HL)
0013'   23                   +          INC     HL
0014'   56                   +          LD      D,(HL)
0015'   C9                   +          RET
0016'   33                   +  MACHIN: INC     SP
0017'   33                   +          INC     SP
0018'   EB                   +          EX      DE,HL
0019'   E9                   +          JP      (HL)            ;ｲﾝﾀPC ｦ ｼﾞｯｺｳ
001A'                        +  ICODE:
                                        ADDQ    DDD,AAA,CCC
001A'   0026'                +          DW      ADDQ
001C'   100C 1000            +          DW      DDD,AAA,CCC
0020'   1008                 +
0022'   0011' 0061'          +          DW      JMP,..0000
0026'   EB                   +          EX      DE,HL
0027'   5E                   +          LD      E,(HL)
0028'   23                   +          INC     HL
0029'   56                   +          LD      D,(HL)
002A'   23                   +          INC     HL
002B'   4E                   +          LD      C,(HL)
002C'   23                   +          INC     HL
002D'   46                   +          LD      B,(HL)
```

```
002E'   23          +               INC     HL
002F'   E5          +               PUSH    HL          ;P ホ゜インタ セーフ゛
0030'   EB          +               EX      DE,HL
0031'   C5          +               PUSH    BC
0032'   DD E1       +               POP     IX
0034'   5E          +               LD      E,(HL)
0035'   23          +               INC     HL
0036'   56          +               LD      D,(HL)
0037'   23          +               INC     HL
0038'   4E          +               LD      C,(HL)
0039'   23          +               INC     HL
003A'   46          +               LD      B,(HL)
003B'   C5          +               PUSH    BC
003C'   D5          +               PUSH    DE
003D'   DD E3       +               EX      (SP),IX
003F'   E1          +               POP     HL
0040'   5E          +               LD      E,(HL)
0041'   23          +               INC     HL
0042'   56          +               LD      D,(HL)
0043'   23          +               INC     HL
0044'   4E          +               LD      C,(HL)
0045'   23          +               INC     HL
0046'   46          +               LD      B,(HL)
0047'   E1          +               POP     HL
0048'   DD 19       +               ADD     IX,DE
004A'   ED 4A       +               ADC     HL,BC
004C'   4D          +               LD      C,L
004D'   44          +               LD      B,H
004E'   E1          +               POP     HL
004F'   5E          +               LD      E,(HL)
0050'   23          +               INC     HL
0051'   56          +               LD      D,(HL)
0052'   23          +               INC     HL
0053'   E5          +               PUSH    HL          ;リタン アト゛レス
0054'   EB          +               EX      DE,HL       ;コタエ ホ゜インタ
0055'   DD E5       +               PUSH    IX
0057'   D1          +               POP     DE
0058'   73          +               LD      (HL),E
0059'   23          +               INC     HL
005A'   72          +               LD      (HL),D
005B'   23          +               INC     HL
005C'   71          +               LD      (HL),C
005D'   23          +               INC     HL
005E'   70          +               LD      (HL),B
005F'   D1          +               POP     DE          ;インタ ホ゜インタ
0060'   C9          +               RET
0061'               +       ..0000:
                                    ADDQ    BBB,JJJ,KKK
0061'   0026'       +               DW      ADDQ
0063'   1004 1014   +               DW      BBB,JJJ,KKK
0067'   1018        +
                                    MACHIN
0069'   0016'       +               DW      MACHIN
006B'   3E 05                       LD      A,5
006D'   32 1010                     LD      (EEE),A
0070'   CD 000E'                    CALL    TO_INTPRI   ;インタフ゜リタヘ
                                    ON      <RED,WED>
```

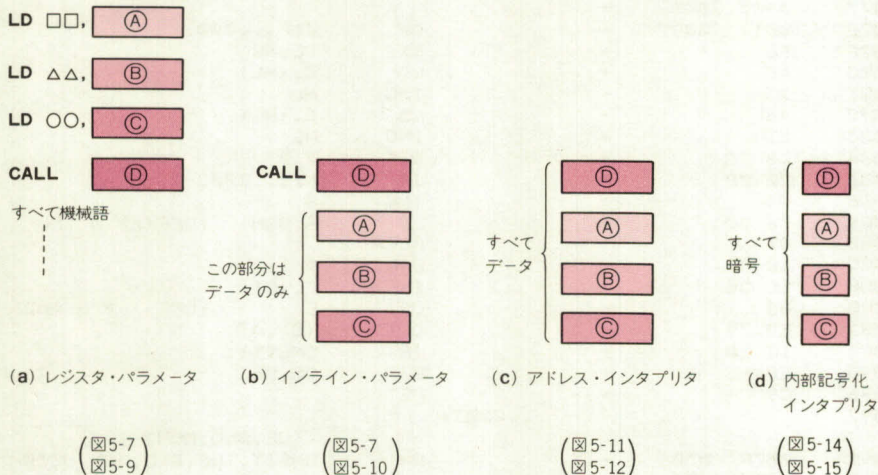
```
0073'   007F' 0108   +              DW      ONBIT,RED,WED,8000H
0077'   0405 8000    +
007B'   0011' 0097'  +              DW      JMP,..0002
007F'   EB           +              EX      DE,HL
0080'   4E           +              LD      C,(HL)
0081'   23           +              INC     HL
0082'   46           +              LD      B,(HL)
0083'   23           +              INC     HL
0084'   CB 78        +              BIT     7,B
0086'   20 0D        +              JR      NZ,..0002-2
0088'   04           +              INC     B
0089'   3E 80        +              LD      A,80H    ;OFFノトキ ハ 7FH
008B'   07           +              RLCA
008C'   10 FD        +              DJNZ    $-1
008E'   ED 58        +              IN      E,(C)
0090'   B3           +              OR      E        ;OFF ノトキ ハ AND
0091'   ED 79        +              OUT     (C),A
0093'   18 EB        +              JR      ONBIT+1
0095'   EB           +              EX      DE,HL
0096'   C9           +              RET
0097'                +      ..0002:
                                    ON      <TUE,BLU,MON>
0097'   007F' 0305   +              DW      ONBIT,TUE,BLU,MON,8000H
009B'   0408 0205    +
009F'   8000         +

                                    JMP     ICODE    ;INTPRIニ フクマレテ イル
00A1'   0011' 001A'  +              DW      JMP,ICODE
                            ;
                                    END     START
```

5.3.5 内部記号化インタプリタ

図5.13を見てください．これまで説明してきた各方式のコード配置を半図式化したものです．この図で(a)，(b)，(c)の各過程は必要最小限の情報以外は生成しないという過程です．(c)ではサブルーチンとデータのアドレスだけが残って普通にいえば余分な情報は含まれていません．それを内部コード・インタプリタでは各1バイトに減らします．

そのためには，サブルーチンや **QUAD** や **PBN** のアドレス表をアセンブル過程で作り，実行時にインタプリタがn番目に登録された **QUAD** はnn(16ビット)番地から始まっているという方式で実アドレスに戻します．サブルーチンも同じで，何番目に登録されたマクロの処理アドレスは何番地からか表を引きます．

この方式によるマクロ呼び出しは図5.14のように，またマクロ定義は図5.15のようになります．図5.14は図5.11とほとんど同じですが，20行のみが少し違います．この中では **AAA** や **BBB** は，もはやアドレスを表していないからです．データ・エリヤの内部は図5.13の(a)〜(d)ですべて全く同じになっていますが，図5.14での **AAA** などの意味がその

〈図5.13〉 各種展開方式における**ADDQ**のメモリ内のコード配置

〈図5.14〉 内部記号化によるインタプリタ

```
 1:  ;  ﾏｸﾛ |ADDQ,ON,JMP,MACHIN|  ﾃｽﾄ
 2:  ;  ｲﾝﾀ ｺｰﾄﾞ ﾆ ﾖﾙ ｲﾝﾀﾌﾟﾘﾀ
 3:  ;
 4:               .XLIST
 5:               INCLUDE B:G1.MAC
 6:               INCLUDE B:G6.MAC ← 図5.15(a)
 7:               .LIST
 8:               ASEG
 9:               ORG        1000H
10:     QUAD      <AAA,BBB,CCC,DDD,EEE>
11:     QUAD      <JJJ,KKK>
12:     PBNDEF    5,<,,MON,TUE,WED>
13:     PBNDEF    8,<,RED,GRN,,BLU,BLK>
14:  ;
15:               INTPRI               ;ｲﾝﾀﾌﾟﾘﾀ ﾖﾋﾞﾀﾞｼ
16:               ADDQ       DDD,AAA,CCC
17:               ADDQ       BBB,JJJ,KKK
18:               MACHIN
19:               LD         A,5
20:               LD         (1020H),A
21:               CALL       TO_INTPRI   ;ｲﾝﾀﾌﾟﾘﾀﾍ
22:               ON         <RED,WED>
23:               ON         <TUE,BLU,MON>
24:               JMP        ICODE     ;|INTPRI|ﾆ ﾌｸﾏﾚﾃ ｲﾙ
25:  ;
26:               END        START
```

〈図5.15〉インタコードを追加したマクロ定義

(a) ソース・リスト

```
 1:   ; |1|ﾊﾞｲﾄ ﾉ ｲﾝﾀｺｰﾄﾞ ﾆ ﾖﾘ
 2:   ; ｼｮﾘ､ﾃﾞｰﾀ ｦ ｱﾗﾜｽ
 3:   ;
 4:   CHECK   MACRO   MNAME,EM
 5:           IF      20H AND NOT TYPE MNAME
 6:   MNAME   EQU     $+3
 7:           ENDIF
 8:           IF      MNAME EQ $+3
 9:           DB      (JMP-SHR_BASE)/2 ;;ｲﾝﾀｺｰﾄﾞ ﾉ JMP
10:           DW      EM
11:           ENDM
12:   ;
13:   ; ｲﾝﾀ|PC|ｶﾗ ｲﾝﾀｺｰﾄﾞ ｦ ﾄﾘﾀﾞｼ
14:   ; ｱﾄﾞﾚｽ ｦ ｹｲｻﾝ ｽﾙ
15:   GETA    MACRO   BASE
16:           LOCAL   EM
17:           LD      HL,BASE&_BASE
18:           CALL    GETA
19:           IF      20H AND NOT TYPE GETA
20:   GETA    EQU     $+2
21:           ENDIF
22:           IF      GETA EQ $+2
23:           JR      EM
24:           PUSH    DE
25:           PUSH    AF
26:           LD      DE,(INTRPC) ;ｲﾝﾀ|PC|
27:           LD      A,(DE)
28:           INC     DE
29:           LD      (INTRPC),DE
30:           ADD     A,A        ;ｵﾌｾｯﾄ
31:           LD      E,A
32:           LD      D,0
33:           RL      D          ;|DE|ﾆ ｵﾌｾｯﾄ
34:           ADD     HL,DE      ;|HL|ﾆ ﾃｰﾌﾞﾙ ｱﾄﾞﾚｽ
35:           LDIRR   E,D
36:           EX      DE,HL      ;|HL|ﾆ ｱﾄﾞﾚｽ
37:           POP     AF
38:           POP     DE
39:           RET
40:           ASEG
41:   INTRPC: DS      2
42:           CSEG
43:   EM:
44:           ENDIF
45:           ENDM
46:   ;
```

```
47:  ADDQ     MACRO    S1,S2,DST
48:           LOCAL    EM
49:           DB (ADDQ-SHR_BASE)/2  ;;ADDQ ショリ ノ ナイブ" コード"
50:           DB (S1-QUAD_BASE)/2   ;;QUAD デ"ータ ノ ナイブ" コード"
51:           DB (S2-QUAD_BASE)/2   ;; "
52:           DB (DST-QUAD_BASE)/2  ;; "
53:           CHECK    ADDQSR,EM
54:           DSEG
55:  ADDQ:    DW       ADDQSR
56:           CSEG
57:           GETA     QUAD   ;インタコード" カラ アト"レス ニ ヘンカン
58:           LDIRR    EDC,B
59:           PUSH     BC
60:           PUSH     DE
61:           GETA     QUAD
62:           LDIRR    EDC,B
63:           POP      IX
64:           POP      HL
65:           ADD      IX,DE    ;ショリ
66:           ADC      HL,BC
67:           PUSH     IX
68:           LD       B,H
69:           LD       C,L
70:           POP      DE
71:           GETA     QUAD
72:           STIR     EDC,B
73:           RET
74:  EM:
75:           ENDIF
76:           ENDM
77:  ;
78:  ON       MACRO    PBNS
79:           LOCAL    EM
80:           DB       (ONBIT-SHR_BASE)/2
81:           IRP      QQ,<PBNS>
82:           DB       (QQ-PBN_BASE)/2
83:           ENDM
84:           DB       -1          ;;エント" コード"
85:           CHECK    ONSR,EM
86:           DSEG
87:  ONBIT:   DW       ONSR
88:           CSEG
89:           LD       DE,(INTRPC)
90:           LD       A,(DE)
91:           INC      A
92:           JR       Z,EM-6   ;エント" コード"
93:           GETA     PBN ;ココデ"ハ アト"レス デ"ナク |PBN|ソノモノ
94:           LD       B,H
95:           LD       C,L
```

```
 96:            INC     B
 97:            LD      A,80H    ;|OFF|ﾉﾄｷ ﾊ |7FH
 98:            RLCA
 99:            DJNZ    $-1
100:            IN      E,(C)
101:            OR      E        ;|OFF| ﾉﾄｷ ﾊ |AND
102:            OUT     (C),A
103:            JR      ONSR
104:            RET
105:            INC     DE
106:            LD      (INTRPC),DE
107:            RET
108:    EM:
109:            ENDIF
110:            ENDM
111:    ;
112:    MACHIN  MACRO            ;ｺｺﾖﾘ ｷｶｲｺﾞ ﾒｲﾚｲ
113:            DB      (MACHIN-SHR_BASE)/2
114:            ENDM
115:    ;
116:    INTPRI  MACRO
117:            LOCAL   JPSHR,MACSHR
118:            DSEG
119:    SHR_BASE:
120:    JMP:    DW      JPSHR
121:    MACHIN: DW      MACSHR
122:            CSEG
123:    START:  LD      HL,ICODE  ;ｲﾝﾀｺｰﾄﾞ ﾊｼﾞﾒ
124:            LD      (INTRPC),HL
125:            LD      BC,$     ;ｼｮﾘ ｶﾗﾉ ｶｴﾘ ﾊ ｺｺ
126:            PUSH    BC
127:            GETA    SHR      ;ｼｮﾘ ｱﾄﾞﾚｽ ｦ ﾓﾄﾒﾙ
128:            JP      (HL)
129:    TO_INTPRI: POP  HL       ;|pointer recover
130:            JR      START+3
131:    JPSHR:  INC     SP       ;ｼﾞｬﾝﾌﾟ ｼｮﾘ
132:            INC     SP
133:            LD      HL,(INTRPC)
134:            LDIRR   E,D
135:            EX      DE,HL
136:            JR      START+3
137:    MACSHR: INC     SP       ;ｺｺﾖﾘ ｷｶｲｺﾞ ｼｮﾘ
138:            INC     SP
139:            LD      HL,(INTRPC)
140:            JP      (HL)         ;ｲﾝﾀ|PC| ｶﾗ ｼﾞｯｺｳ ｽﾙ
141:    ICODE:
142:            ENDM
143:    ;
144:    JMP     MACRO   ADS
```

```
145:              DB       (JMP-SHR_BASE)/2
146:              DW       ADS
147:              ENDM
148:  ;
149:  PBNDEF      MACRO    PORT,BNAME
150:              DSEG                ;;ポ゚インタ テーブ゙ル ヲ DSEG ニ ツクル
151:              IFNDEF   PBN_BASE  ;;PBNテーブ゙ル ノ センドウ
152:  PBN_BASE:
153:              ENDIF
154:  TEMP        DEFL     0
155:              IRP      Q,<BNAME>
156:              IFNB     <Q>
157:  Q:          DW       TEMP*256+PORT
158:              ENDIF
159:  TEMP        DEFL     TEMP+1
160:              ENDM
161:              ENDM
162:  ;
163:  QUAD        MACRO    NAMES
164:              DSEG               ;; テーブ゙ル ヲ デ゙ータ セグ゙メント ニ
165:              IFNDEF   QUAD_BASE
166:  QUAD_BASE:
167:              ENDIF
168:              IRP      QQ,<NAMES>
169:              ASEG               ;;デ゙ータ エリヤ ヲ RAM ニ ツクル
170:  TEMP        DEFL     $
171:              DS       4          ;;QUAD ダ゙カラ
172:              DSEG
173:  QQ:         DW       TEMP
174:              ENDM
175:              ENDM
176:  ;
```

(b) アセンブル・リスト

```
                                        ; マクロ ADDQ,ON,JMP,MACHIN テスト
                                        ; インタ コード゙ ニ ヨル インタプ゚リタ
                                        ;
                                               .LIST
0000'                                          ASEG
                                               ORG      1000H
                                      QUAD     <AAA,BBB,CCC,DDD,EEE>
1000                      +                    DSEG
0000"                     +                    ASEG
1000                      +                    DS       4
1004                      +                    DSEG
0000"    1000             +           AAA:     DW       TEMP
0002"                     +                    ASEG
1004                      +                    DS       4
```

```
1008                +              DSEG
0002"    1004       +     BBB:     DW       TEMP
0004"               +              ASEG
1008                +              DS       4
100C                +              DSEG
0004"    1008       +     CCC:     DW       TEMP
0006"               +              ASEG
100C                +              DS       4
1010                +              DSEG
0006"    100C       +     DDD:     DW       TEMP
0008"               +              ASEG
1010                +              DS       4
1014                +              DSEG
0008"    1010       +     EEE:     DW       TEMP
                           QUAD    <JJJ,KKK>
000A"               +              DSEG
000A"               +              ASEG
1014                +              DS       4
1018                +              DSEG
000A"    1014       +     JJJ:     DW       TEMP
000C"               +              ASEG
1018                +              DS       4
101C                +              DSEG
000C"    1018       +     KKK:     DW       TEMP
                           PBNDEF  5,<,,MON,TUE,WED>
000E"               +              DSEG
000E"    0205       +     MON:     DW       TEMP*256+5
0010"    0305       +     TUE:     DW       TEMP*256+5
0012"    0405       +     WED:     DW       TEMP*256+5
                           PBNDEF  8,<,RED,GRN,,BLU,BLK>
0014"               +              DSEG
0014"    0108       +     RED:     DW       TEMP*256+8
0016"    0208       +     GRN:     DW       TEMP*256+8
0018"    0408       +     BLU:     DW       TEMP*256+8
001A"    0508       +     BLK:     DW       TEMP*256+8
                          ;
                                   INTPRI                 ;ｲﾝﾀﾌﾟﾘﾀ ﾖﾋﾞﾀﾞｼ
001C"               +              DSEG
001C"               +     SHR_BASE:
001C"    0030'      +     JMP:     DW       ..0000
001E"    003B'      +     MACHIN:  DW       ..0001
0020"               +              CSEG
0000'    21 0041'   +     START:   LD       HL,ICODE   ;ｲﾝﾀｺｰﾄﾞ ﾊｼﾞﾒ
0003'    22 101C    +              LD       (INTRPC),HL
0006'    01 0006'   +              LD       BC,$       ;ｼｮﾘ ｶﾗﾉ ｶｴﾘ ﾊ ｺｺ
0009'    C5         +              PUSH     BC
000A'    21 001C"   +              LD       HL,SHR&_BASE
000D'    CD 0012'   +              CALL     GETA
0010'    18 1A      +              JR       ..0002
0012'    D5         +              PUSH     DE
0013'    F5         +              PUSH     AF
0014'    ED 5B 101C +              LD       DE,(INTRPC) ;ｲﾝﾀPC
0018'    1A         +              LD       A,(DE)
0019'    13         +              INC      DE
001A'    ED 53 101C +              LD       (INTRPC),DE
001E'    87         +              ADD      A,A        ;ｵﾌｾｯﾄ
```

```
001F'   5F          +                LD    E,A
0020'   16 00       +                LD    D,0
0022'   CB 12       +                RL    D          ;DEニ オフセット
0024'   19          +                ADD   HL,DE      ;HLニ テーブル アドレス
0025'   5E          +                LD    E,(HL)
0026'   23          +                INC   HL
0027'   56          +                LD    D,(HL)
0028'   EB          +                EX    DE,HL      ;HLニ アドレス
0029'   F1          +                POP   AF
002A'   D1          +                POP   DE
002B'   C9          +                RET
002C'               +                ASEG
101C                +   INTRPC:      DS    2
101E                +                CSEG
002C'               +   ..0002:
002C'   E9          +                JP    (HL)
002D'   E1          +   TO_INTPRI:   POP   HL         ;pointer recover
002E'   18 D3       +                JR    START+3
0030'   33          +   ..0000:      INC   SP         ;ジャンプ ショリ
0031'   33          +                INC   SP
0032'   2A 101C     +                LD    HL,(INTRPC)
0035'   5E          +                LD    E,(HL)
0036'   23          +                INC   HL
0037'   56          +                LD    D,(HL)
0038'   EB          +                EX    DE,HL
0039'   18 C8       +                JR    START+3
003B'   33          +   ..0001:      INC   SP         ;ココヨリ キカイゴ ショリ
003C'   33          +                INC   SP
003D'   2A 101C     +                LD    HL,(INTRPC)
0040'   E9          +                JP    (HL)        ;インタPC カラ ジッコウ
0041'               +   ICODE:
                                     ADDQ  DDD,AAA,CCC
0041'   02          +                DB (ADDQ-SHR_BASE)/2
0042'   03          +                DB (DDD-QUAD_BASE)/2
0043'   00          +                DB (AAA-QUAD_BASE)/2
0044'   02          +                DB (CCC-QUAD_BASE)/2
0045'   00          +                DB    (JMP-SHR_BASE)/2
0046'   007E'       +                DW    ..0003
0048'               +                DSEG
0020"   0048'       +   ADDQ:        DW    ADDQSR
0022"               +                CSEG
0048'   21 0000"    +                LD    HL,QUAD&_BASE
004B'   CD 0012'    +                CALL  GETA
004E'   5E          +                LD    E,(HL)
004F'   23          +                INC   HL
0050'   56          +                LD    D,(HL)
0051'   23          +                INC   HL
0052'   4E          +                LD    C,(HL)
0053'   23          +                INC   HL
0054'   46          +                LD    B,(HL)
0055'   C5          +                PUSH  BC
0056'   D5          +                PUSH  DE
0057'   21 0000"    +                LD    HL,QUAD&_BASE
005A'   CD 0012'    +                CALL  GETA
```

```
005D'   5E             +              LD      E,(HL)
005E'   23             +              INC     HL
005F'   56             +              LD      D,(HL)
0060'   23             +              INC     HL
0061'   4E             +              LD      C,(HL)
0062'   23             +              INC     HL
0063'   46             +              LD      B,(HL)
0064'   DD E1          +              POP     IX
0066'   E1             +              POP     HL
0067'   DD 19          +              ADD     IX,DE    ;ｼｮﾘ
0069'   ED 4A          +              ADC     HL,BC
006B'   DD E5          +              PUSH    IX
006D'   44             +              LD      B,H
006E'   4D             +              LD      C,L
006F'   D1             +              POP     DE
0070'   21 0000"       +              LD      HL,QUAD&_BASE
0073'   CD 0012'       +              CALL    GETA
0076'   73             +              LD      (HL),E
0077'   23             +              INC     HL
0078'   72             +              LD      (HL),D
0079'   23             +              INC     HL
007A'   71             +              LD      (HL),C
007B'   23             +              INC     HL
007C'   70             +              LD      (HL),B
007D'   C9             +              RET
007E'                  +      ..0003:
                                      ADDQ    BBB,JJJ,KKK
007E'   02             +              DB (ADDQ-SHR_BASE)/2
007F'   01             +              DB (BBB-QUAD_BASE)/2
0080'   05             +              DB (JJJ-QUAD_BASE)/2
0081'   06             +              DB (KKK-QUAD_BASE)/2
                                      MACHIN
0082'   01             +              DB      (MACHIN-SHR_BASE)/2
0083'   3E 05                         LD      A,5
0085'   32 1020                       LD      (1020H),A
0088'   CD 002D'                      CALL    TO_INTPRI    ;ｲﾝﾀﾌﾟﾘﾀﾍ
                                      ON      <RED,WED>
008B'   03             +              DB      (ONBIT-SHR_BASE)/2
008C'   03             +              DB      (RED-PBN_BASE)/2
008D'   02             +              DB      (WED-PBN_BASE)/2
008E'   FF             +              DB      -1
008F'   00             +              DB      (JMP-SHR_BASE)/2
0090'   00B6'          +              DW      ..0008
0092'                  +              DSEG
0022"   0092'          +      ONBIT:  DW      ONSR
0024"                  +              CSEG
0092'   ED 5B 101C     +              LD      DE,(INTRPC)
0096'   1A             +              LD      A,(DE)
0097'   3C             +              INC     A
0098'   28 16          +              JR      Z,..0008-6        ;ｴﾝﾄﾞ ｺｰﾄﾞ
009A'   21 000E"       +              LD      HL,PBN&_BASE
009D'   CD 0012'       +              CALL    GETA
00A0'   44             +              LD      B,H
00A1'   4D             +              LD      C,L
00A2'   04             +              INC     B
```

```
00A3'   3E 80       +               LD      A,80H     ;OFFノトキ ハ 7FH
00A5'   07          +               RLCA
00A6'   10 FD       +               DJNZ    $-1
00A8'   ED 58       +               IN      E,(C)
00AA'   B3          +               OR      E         ;OFF ノトキ ハ AND
00AB'   ED 79       +               OUT     (C),A
00AD'   18 E3       +               JR      ONSR
00AF'   C9          +               RET
00B0'   13          +               INC     DE
00B1'   ED 53 101C  +               LD      (INTRPC),DE
00B5'   C9          +               RET
00B6'               +       ..0008:
                                    ON      <TUE,BLU,MON>
00B6'   03          +               DB      (ONBIT-SHR_BASE)/2
00B7'   01          +               DB      (TUE-PBN_BASE)/2
00B8'   05          +               DB      (BLU-PBN_BASE)/2
00B9'   00          +               DB      (MON-PBN_BASE)/2
00BA'   FF          +               DB      -1
                                    JMP     ICODE     ;INTPRIニ フクマレテ イル
00BB'   00          +               DB      (JMP-SHR_BASE)/2
00BC'   0041'       +               DW      ICODE
                            ;
                                    END     START
```

番地を直接表している数値でなく，登録表のアドレスになっているのです．

処理アドレスとデータの種類を8ビットで表す関係上，同一種類のデータは255個までしか登録できません．また，マクロから呼び出すサブルーチンも255個までです．それでも通常の処理で不足することはありません．データがファイルのような大きなものになれば，その個々に名前を付けずポインタで参照すればよいわけで，データ個数が各々255で不足するケースは避けた方が良いプログラムになるともいえます．

MACHIN や **JMP** は前項と全く同じ使い方です．**MACHIN** の後の機械語部分では，**CALL**，**RET** も含めてすべての機械語が使えます．何処へ **JP** しても，その位置で **CALL TO_INTPRI** を実行すれば，その次のアドレスからインタコードとして実行されます．

インタプリタの構造は本質的には前項と同じですが，すべてのアドレスを表から引いてくるので，そのための専用サブルーチン **GETA** が用意されています．**GETA** は通常 **INTPRI** の中に展開されます．

それでは，インタプリタの各命令はどの時点でメモリの中に配置されるのでしょうか．通常はインタプリタといえばすべての実行ルーチンをあらかじめ内蔵しているものです．アセンブラで書いても通常はそのようにします．しかし，ここで示している前項と本項のインタプリタは，共に呼び出されたマクロを処理するためのルーチンのみしか生成しません．したがって，プログラム・メモリを極限まで小さくすることができます．

また，内部コードからアドレスを引く場合は，その表をどこかに作っておかなければなりません．ところが，呼び出されたルーチンだけを作成するとなると，最初に表を作ろうとしてもその表のデータ内容はもちろん，その表が何バイト(アドレス何個分)必要になるかも決まりません．もし，各々のデータとサブルーチン用に255個分(510バイト)のメモリを表のために取っておくとすれば，メモリは大きな無駄を生じることになります．

この問題はアドレス参照のための表を **DSEG** 内に作るようにして解決しています．このプログラムではデータ用RAMエリヤを **ASEG** とし，アドレス表専用に **DSEG** を使用しています．リンク時に **/D**：を指定しなければ **DSEG** と **CSEG** の間にスキ間はなく，メモリは1バイトも無駄になりません．

QUAD や **PBNDEF** も前項までとは全く違う内部コード用の値とその中に入れておくアドレスとを定義するようになっています．このあたりの展開作業については，図5.15(b)をよく見ればわかります．

この方式で，結局 **ADDQ** で4バイトにまで減りました．直接データを渡す方式と比べて6分の1以下です．実行ルーチンを考えても，60Kのプログラムが10Kになるくらいと考えられます．スピードは遅くなりますが記述性は全く変わらず，マクロ定義のみでこれだけの変化が出せるということは注目に値すると思います．

5.4 高級言語風マクロ表記

```
LET  AA  =  BB  +  CC
```

このように書けば，その意味を説明する必要はないでしょう．こんな書き方がマクロ・アセンブラでもできるとすれば素晴らしいと思われるでしょう．これが実際にできます．M80ではこの表記を解読できます．変数名と記号の間を詰めて書くことはできませんが，ブランクで区切れば問題はありません．

5.4.1 1バイト八則演算

八則というのは，四則の他に **AND**，**OR**，**XOR**，**BIT CLEAR** の四則を加えたものです．**BIT CREAR** は論理引算のようなもので，**AND NOT** と同じ意味です．高級言語風ということで，論理の四則も1文字の演算子で表現します．上記の順に **&**，¦，〜，**¥** の記号を使います．

このマクロの定義は図5.16のようになります．ここでは**LET**とせず，**.1**としています．1バイト演算を示すためです．また＝は使用せず，必要なパラメータのみを書くようにしました．**.1**の中に代入せよという意味を感じれば＝を書く必要はありません．筆者の考えでは書く文字(打つ文字)は1文字でも少ない方がよいと思い，このような書式で使用しています．

ここでは1バイトの八則としましたが，2バイトまでは簡単に拡張できると思います．

〈図5.16〉 1バイト八則演算のマクロ定義

(a) ソース・リスト

```
 1:                  .Z80
 2:   ; コーキューゲ゛ンコ゛ フー マクロ
 3:   ;
 4:                  .XLIST
 5:   .1             MACRO     P1,P2,P3,P4
 6:                  LD        A,(P2)
 7:                  LD        HL,P4
 8:                  IFIDN     <P3>,<+>
 9:                  ADD       A,(HL)
10:                  ENDIF
11:                  IFIDN     <P3>,<->
12:                  SUB       (HL)
13:                  ENDIF
14:                  IFIDN     <P3>,<*>
15:                  CALL      MUL
16:                  ENDIF
17:                  IFIDN     <P3>,</>
18:                  CALL      DIV
19:                  ENDIF
20:                  IFIDN     <P3>,<&> ;!AND
21:                  AND       (HL)
22:                  ENDIF
23:                  IFIDN     <P3>,<!> ;!OR
24:                  OR        (HL)
25:                  ENDIF
26:                  IFIDN     <P3>,<~> ;!EXOR
27:                  XOR       (HL)
28:                  ENDIF
29:                  IFIDN     <P3>,<¥> ;!BIT CLEAR
30:                  CPL
31:                  OR        (HL)
32:                  CPL
33:                  ENDIF
34:                  LD        (P1),A
```

```
35:             ENDM
36:             .LIST
37:     ;
38:     ; マクロ ショリヨウ サブ゛ルーチン
39:     ;
40:     MUL:    LD      B,(HL)  ;カケザ゛ン
41:             LD      C,A
42:             DEC     B
43:             ADD     A,C
44:             DJNZ    $-1
45:             RET
46:     DIV:    LD      C,(HL)  ;ワリザ゛ン
47:             LD      B,-1
48:             INC     B
49:             SUB     C
50:             JR      NC,$-2
51:             LD      A,B
52:             RET
53:     ;
54:     ; コーキューゲ゛ンゴ゛ フー マクロ ヨビ゛ダ゛シ
55:     ;
56:     AA:     DB      1
57:     BB:     DB      2
58:     CC:     DB      3
59:     DD:     DB      4
60:     EE:     DB      5
61:     ;
62:     FF:     DS      1
63:     GG:     DS      1
64:     HH:     DS      1
65:     ;
66:             .1      GG AA + CC
67:             .1      FF BB - CC
68:             .1      GG DD * EE
69:             .1      HH EE / DD
70:             .1      FF AA & BB
71:             .1      GG EE ! CC
72:             .1      HH DD ~ BB
73:             .1      GG AA ¥ DD
74:             END
```

(b) アセンブル・リスト

```
                        .Z80
                ; コーキューゲ゛ンゴ゛ フー マクロ
                ;
                        .LIST
                ;
                ; マクロ ショリヨウ サブ゛ルーチン
                ;
0000'   46      MUL:    LD      B,(HL)  ;カケザ゛ン
0001'   4F              LD      C,A
```

```
0002'   05                      DEC     B
0003'   81                      ADD     A,C
0004'   10 FD                   DJNZ    $-1
0006'   C9                      RET
0007'   4E              DIV:    LD      C,(HL)  ;ﾜﾘｻﾞﾝ
0008'   06 FF                   LD      B,-1
000A'   04                      INC     B
000B'   91                      SUB     C
000C'   30 FC                   JR      NC,$-2
000E'   78                      LD      A,B
000F'   C9                      RET
                        ;
                        ; ｺｰｷｭｰｹﾞﾝｺﾞ ﾌｰ ﾏｸﾛ ﾖﾋﾞﾀﾞｼ
                        ;
0010'   01              AA:     DB      1
0011'   02              BB:     DB      2
0012'   03              CC:     DB      3
0013'   04              DD:     DB      4
0014'   05              EE:     DB      5
                        ;
0015'                   FF:     DS      1
0016'                   GG:     DS      1
0017'                   HH:     DS      1
                        ;
                                .1      GG AA + CC
0018'   3A 0010'        +       LD      A,(AA)
001B'   21 0012'        +       LD      HL,CC
001E'   86              +       ADD     A,(HL)
001F'   32 0016'        +       LD      (GG),A
                                .1      FF BB - CC
0022'   3A 0011'        +       LD      A,(BB)
0025'   21 0012'        +       LD      HL,CC
0028'   96              +       SUB     (HL)
0029'   32 0015'        +       LD      (FF),A
                                .1      GG DD * EE
002C'   3A 0013'        +       LD      A,(DD)
002F'   21 0014'        +       LD      HL,EE
0032'   CD 0000'        +       CALL    MUL
0035'   32 0016'        +       LD      (GG),A
                                .1      HH EE / DD
0038'   3A 0014'        +       LD      A,(EE)
003B'   21 0013'        +       LD      HL,DD
003E'   CD 0007'        +       CALL    DIV
0041'   32 0017'        +       LD      (HH),A
                                .1      FF AA & BB
0044'   3A 0010'        +       LD      A,(AA)
0047'   21 0011'        +       LD      HL,BB
004A'   A6              +       AND     (HL)
004B'   32 0015'        +       LD      (FF),A
                                .1      GG EE ! CC
004E'   3A 0014'        +       LD      A,(EE)
0051'   21 0012'        +       LD      HL,CC
0054'   B6              +       OR      (HL)
0055'   32 0016'        +       LD      (GG),A
```

```
                                    .1      HH DD ~ BB
0058'   3A 0013'        +           LD      A,(DD)
005B'   21 0011'        +           LD      HL,BB
005E'   AE              +           XOR     (HL)
005F'   32 0017'        +           LD      (HH),A
                                    .1      GG AA ¥ DD
0062'   3A 0010'        +           LD      A,(AA)
0065'   21 0013'        +           LD      HL,DD
0068'   2F              +           CPL
0069'   B6              +           OR      (HL)
006A'   2F              +           CPL
006B'   32 0016'        +           LD      (GG),A
                                    END
```

5.4.2 4バイト四則演算

4バイトとなると少し手強いですが，加減算までは何とかなります．図5.17のようなマクロ定義になります．呼び出しフォーマットは，**.4**として前項と同じ形式になっています．どうしても**LET**でなければという場合には図5.17(c)のように書けばよいでしょう．：＝は＝だけでもかまいません．書かなければ誤動作します．

図5.17は乗除算のマクロを含んでいません．これは図5.18を取り込んでいます．このマクロ・ファイルには乗除算の他にルートやBCD変換が含まれています．参考にしてください．

高級言語風表記といっても，第3パラメータの文字を**IFIDN**でチェックしているだけですからたいしたことはありません．しかしながら，実際にこの書式でアセンブラ内に書いてあると読みやすさは抜群でコーディング・ミスは激減します．マクロ定義を見ればわかるとおり，カッコや連続演算はできませんが，それでも十分に効果があります．アセンブラで算術演算を多用する時はぜひ試してください．そして一度試せば必ず必需品になってしまうでしょう．

5.4.3 4バイト四則演算インタプリタ

表記法を図5.17(c)と同じにして，展開法を5.3.4項と同じアドレス・リストによるインタプリタとしたのが図5.19です．乗除算についてはデータ部分を読み捨てる機能しか書いてありませんが，図5.18を応用すれば簡単にできます．インタプリタの実行手順は5.3.4項と全く同じです．ただし，インタプリタのルーチンや4バイト・データの定義をマクロ化していない点だけは違います．

このように，マクロ表記の方法と，マクロ展開の方法とはお互いに独立しており，この書き方ではこのように展開しなければいけないという制約は全くありません．その都度最適な表現法，最適な展開法を任意に組み合わせて使用できます．

〈図5.17〉 4バイト四則演算のマクロ定義

(a) ソース・リスト

```
 1:         .Z80
 2:  ; コーキューゲンゴ フー マクロ
 3:  ; |4|バイト エンザン
 4:  ;
 5:         .XLIST
 6:         INCLUDE  B:A17.MAC ←図5.18マクロ定義を取り込む
 7:  ;
 8:  ; シキ ノ カイセキ
 9:  ;
10:  .4     MACRO    P1,P2,P3,P4
11:         IFIDN    <P3>,<+>
12:         LD       HL,(P2)
13:         LD       DE,(P4)
14:         ADD      HL,DE
15:         LD       (P1),HL
16:         LD       HL,(P2+2)
17:         LD       DE,(P4+2)
18:         ADC      HL,DE
19:         LD       (P1+2),HL
20:         ENDIF
21:         IFIDN    <P3>,<->
22:         LD       HL,(P2)
23:         LD       DE,(P4)
24:         AND      A          ;|clr cari
25:         SBC      HL,DE
26:         LD       (P1),HL
27:         LD       HL,(P2+2)
28:         LD       DE,(P4+2)
29:         SBC      HL,DE
30:         ENDIF
31:         LD       (P1+2),HL
32:         IFIDN    <P3>,<*>
33:         MULQ     P2,P4,P1
34:         ENDIF
35:         IFIDN    <P3>,</>
36:         DIVQ     P2,P4,P1
37:         ENDIF
38:         ENDM
39:         .LIST
40:  ;
41:  AA:    DW       5678H,1234H
42:  BB:    DW       3344H,1122H
43:  CC:    DS       4
44:  MPBF:  DS       8          ;テンポラリ
45:  ;
```

```
46:   ; コーキューゲ゛ンコ゛ フー マクロ ヨビ゛ダ゛シ
47:   ;
48:   ; スベ゛テ ¦4¦バ゛イト¦(32¦ビ゛ット¦)¦ ノ エンザ゛ン
49:   ;
50:            .4      CC AA + BB
51:            .4      CC AA - BB
52:            .4      CC AA * BB
53:            .4      CC AA / BB
54:            .4      CC AA * BB
55:            .4      CC AA / BB
56:   ;
57:            END
```

(b) アセンブリル・リスト

```
                                        .Z80
                                ; コーキューゲ゛ンコ゛ フー マクロ
                                ; 4バ゛イト エンザ゛ン
                                ;
                                        .LIST
                                ;
0000'   5678 1234               AA:     DW      5678H,1234H
0004'   3344 1122               BB:     DW      3344H,1122H
0008'                           CC:     DS      4
000C'                           MPBF:   DS      8       ;テンポ゛ラリ
                                ;
                                ; コーキューゲ゛ンコ゛ フー マクロ ヨビ゛ダ゛シ
                                ;
                                ; スベ゛テ 4バ゛イト(32ビ゛ット) ノ エンザ゛ン
                                ;
                                        .4      CC AA + BB
0014'   2A 0000'          +             LD      HL,(AA)
0017'   ED 5B 0004'       +             LD      DE,(BB)
001B'   19                +             ADD     HL,DE
001C'   22 0008'          +             LD      (CC),HL
001F'   2A 0002'          +             LD      HL,(AA+2)
0022'   ED 5B 0006'       +             LD      DE,(BB+2)
0026'   ED 5A             +             ADC     HL,DE
0028'   22 000A'          +             LD      (CC+2),HL
002B'   22 000A'          +             LD      (CC+2),HL
                                        .4      CC AA - BB
002E'   2A 0000'          +             LD      HL,(AA)
0031'   ED 5B 0004'       +             LD      DE,(BB)
0035'   A7                +             AND     A       ;clr cari
0036'   ED 52             +             SBC     HL,DE
0038'   22 0008'          +             LD      (CC),HL
003B'   2A 0002'          +             LD      HL,(AA+2)
003E'   ED 5B 0006'       +             LD      DE,(BB+2)
0042'   ED 52             +             SBC     HL,DE
0044'   22 000A'          +             LD      (CC+2),HL
                                        .4      CC AA * BB
0047'   22 000A'          +             LD      (CC+2),HL
004A'   21 0000'          +             LD      HL,AA
004D'   DD 21 0004'       +             LD      IX,BB
0051'   CD 0065'          +             CALL    MULQ
0054'   EB                +             EX      DE,HL
0055'   21 0008'          +             LD      HL,CC
```

```
0058'   3A 000F'      +            LD      A,(MPBF+3)
005B'   77            +            LD      (HL),A
005C'   23            +            INC     HL
005D'   73            +            LD      (HL),E
005E'   23            +            INC     HL
005F'   72            +            LD      (HL),D
0060'   23            +            INC     HL
0061'   71            +            LD      (HL),C
0062'   C3 00C2'      +            JP      ..0000
0065'   FD 21 000C'   +            LD      IY,MPBF
0069'   11 000C'      +            LD      DE,MPBF
006C'   01 0004       +            LD      BC,4
006F'   ED B0         +            LDIR
0071'   DD 5E 00      +            LD      E,(IX)
0074'   DD 56 01      +            LD      D,(IX+1)
0077'   DD 4E 02      +            LD      C,(IX+2)
007A'   DD 46 03      +            LD      B,(IX+3)
007D'   21 0000       +            LD      HL,0
0080'   E5            +            PUSH    HL
0081'   21 0000       +            LD      HL,0
0084'   3E 21         +            LD      A,33       ;ｶｳﾝﾀ
0086'   FD CB 03 3E   +            SRL     (IY+3)
008A'   FD CB 02 1E   +   ..0001:  RR      (IY+2)
008E'   FD CB 01 1E   +            RR      (IY+1)
0092'   FD CB 00 1E   +            RR      (IY)
0096'   3D            +            DEC     A
0097'   28 18         +            JR      Z,..0004
0099'   30 13         +            JR      NC,..0003
009B'   19            +            ADD     HL,DE
009C'   E3            +            EX      (SP),HL
009D'   ED 4A         +            ADC     HL,BC
009F'   CB 1C         +   ..0002:  RR      H
00A1'   CB 1D         +            RR      L
00A3'   E3            +            EX      (SP),HL
00A4'   CB 1C         +            RR      H
00A6'   CB 1D         +            RR      L
00A8'   FD CB 03 1E   +            RR      (IY+3)
00AC'   18 DC         +            JR      ..0001
00AE'   E3            +   ..0003:  EX      (SP),HL
00AF'   18 EE         +            JR      ..0002
00B1'   C1            +   ..0004:  POP     BC         ;BCHL ﾆ ｺﾀｴ
00B2'   3A 000F'      +            LD      A,(MPBF+3)
00B5'   FD CB 03 7E   +            BIT     7,(IY+3)   ;4/5
00B9'   C8            +            RET     Z          ;4ｼｬ
00BA'   2C            +            INC     L          ;5ﾆｭｳ
00BB'   C0            +            RET     NZ
00BC'   24            +            INC     H
00BD'   C0            +            RET     NZ
00BE'   0C            +            INC     C
00BF'   C0            +            RET     NZ
00C0'   04            +            INC     B
00C1'   C9            +            RET
00C2'                 +   ..0000:
                                   .4      CC AA / BB
00C2'   22 000A'      +            LD      (CC+2),HL
00C5'   21 0000'      +            LD      HL,AA
```

```
00C8'   DD 21 0004'    +           LD      IX,BB
00CC'   CD 00D9'       +           CALL    DIVQ
00CF'   22 0008'       +           LD      (CC),HL ;DEHL ニ コタエ
00D2'   ED 53 000A'    +           LD      (CC+2),DE
00D6'   C3 013E'       +           JP      ..0005
00D9'   AF             +           XOR     A
00DA'   32 000E'       +           LD      (MPBF+2),A
00DD'   47             +           LD      B,A
00DE'   7E             +           LD      A,(HL)
00DF'   23             +           INC     HL
00E0'   5E             +           LD      E,(HL)
00E1'   23             +           INC     HL
00E2'   56             +           LD      D,(HL)
00E3'   23             +           INC     HL
00E4'   4E             +           LD      C,(HL)
00E5'   C5             +           PUSH    BC
00E6'   21 0000        +           LD      HL,0
00E9'   22 000C'       +           LD      (MPBF),HL
00EC'   EB             +           EX      DE,HL
00ED'   DD 5E 00       +           LD      E,(IX)
00F0'   DD 56 01       +           LD      D,(IX+1)
00F3'   DD 4E 02       +           LD      C,(IX+2)
00F6'   DD 46 03       +           LD      B,(IX+3)
00F9'   FD 21 000C'    +           LD      IY,MPBF
00FD'   3E 22          +           LD      A,34     ;Acc カウンタ
00FF'   A7             +  ..0006:  AND     A
0100'   ED 52          +           SBC     HL,DE
0102'   E3             +           EX      (SP),HL
0103'   ED 42          +           SBC     HL,BC    ;HL'
0105'   3D             +           DEC     A
0106'   28 25          +           JR      Z,..0008
0108'   30 05          +           JR      NC,..0007
010A'   E3             +           EX      (SP),HL
010B'   19             +           ADD     HL,DE
010C'   E3             +           EX      (SP),HL
010D'   ED 4A          +           ADC     HL,BC    ;HL'
010F'   3F             +  ..0007:  CCF
0110'   E3             +           EX      (SP),HL
0111'   FD CB 00 16    +           RL      (IY)
0115'   FD CB 01 16    +           RL      (IY+1)
0119'   FD CB 02 16    +           RL      (IY+2)
011D'   FD CB 03 16    +           RL      (IY+3)
0121'   CB 15          +           RL      L
0123'   CB 14          +           RL      H
0125'   E3             +           EX      (SP),HL
0126'   CB 15          +           RL      L
0128'   CB 14          +           RL      H        ;HL'
012A'   E3             +           EX      (SP),HL
012B'   18 D2          +           JR      ..0006
012D'   C1             +  ..0008:  POP     BC
012E'   ED 5B 000E'    +           LD      DE,(MPBF+2)
0132'   2A 000C'       +           LD      HL,(MPBF)
0135'   D8             +           RET     C        ;4シャ
0136'   2C             +           INC     L        ;5ニュウ
0137'   C0             +           RET     NZ
0138'   24             +           INC     H
0139'   C0             +           RET     NZ
```

```
013A'   1C              +       INC     E
013B'   C0              +       RET     NZ
013C'   14              +       INC     D
013D'   C9              +       RET
013E'                   +  ..0005:
                                .4      CC AA * BB
013E'   22 000A'        +       LD      (CC+2),HL
0141'   21 0000'        +       LD      HL,AA
0144'   DD 21 0004'     +       LD      IX,BB
0148'   CD 0065'        +       CALL    MULQ
014B'   EB              +       EX      DE,HL
014C'   21 0008'        +       LD      HL,CC
014F'   3A 000F'        +       LD      A,(MPBF+3)
0152'   77              +       LD      (HL),A
0153'   23              +       INC     HL
0154'   73              +       LD      (HL),E
0155'   23              +       INC     HL
0156'   72              +       LD      (HL),D
0157'   23              +       INC     HL
0158'   71              +       LD      (HL),C
                                .4      CC AA / BB
0159'   22 000A'        +       LD      (CC+2),HL
015C'   21 0000'        +       LD      HL,AA
015F'   DD 21 0004'     +       LD      IX,BB
0163'   CD 00D9'        +       CALL    DIVQ
0166'   22 0008'        +       LD      (CC),HL ;DEHL ニ コタエ
0169'   ED 53 000A'     +       LD      (CC+2),DE
                           ;
                                END
```

(c) **LET**と：＝を使った呼び出し形式

```
 1:             .Z80
 2:     ; コーキューゲンゴ フー マクロ
 3:     ; |4|バイト エンザン
 4:     ;
 5:             .XLIST
 6:             INCLUDE  B:A17.MAC
 7:     ;
 8:     ; シキ ノ カイセキ
 9:     ;
10:     LET     MACRO    P1,DD,P2,P3,P4 ;DD ハ ダミー。 = カ := ヲ カク
11:             IFIDN    <P3>,<+>
12:             LD       HL,(P2)
13:             LD       DE,(P4)
14:             ADD      HL,DE
15:             LD       (P1),HL
16:             LD       HL,(P2+2)
17:             LD       DE,(P4+2)
18:             ADC      HL,DE
19:             LD       (P1+2),HL
20:             ENDIF
21:             IFIDN    <P3>,<->
22:             LD       HL,(P2)
23:             LD       DE,(P4)
```

```
24:            AND     A          ;|clr cari
25:            SBC     HL,DE
26:            LD      (P1),HL
27:            LD      HL,(P2+2)
28:            LD      DE,(P4+2)
29:            SBC     HL,DE
30:            ENDIF
31:            LD      (P1+2),HL
32:            IFIDN   <P3>,<*>
33:            MULQ    P2,P4,P1
34:            ENDIF
35:            IFIDN   <P3>,</>
36:            DIVQ    P2,P4,P1
37:            ENDIF
38:            ENDM
39:            .LIST
40:  ;
41:  AA:       DW      5678H,1234H
42:  BB:       DW      3344H,1122H
43:  CC:       DS      4
44:  MPBF:     DS      8          ;ﾃﾝﾎﾟﾗﾘ
45:  ;
46:  ; ｺｰｷｭｰｹﾞﾝｺﾞ ﾌｰ ﾏｸﾛ ﾖﾋﾞﾀﾞｼ
47:  ;
48:  ; ｽﾍﾞﾃ |4|ﾊﾞｲﾄ|(32|ﾋﾞｯﾄ|)| ﾉ ｴﾝｻﾞﾝ
49:  ;
50:            LET     CC := AA + BB
51:            LET     CC := AA - BB
52:            LET     CC := AA * BB
53:            LET     CC := AA / BB
54:            LET     CC := AA * BB
55:            LET     CC := AA / BB
56:  ;
57:            END
```

高級言語のフィーリングがただよう表記法
:= は = と書いてもよい

〈図 5.18〉 4 バイト演算用マクロ集
（1度のみサブルーチンを作る．6 バイト BCD 変換を含む）

```
 1:  ;ｲﾁﾄﾞ ﾉﾐ ﾃﾝｶｲｽﾙ ﾀﾒﾉ ﾁｪｯｸ
 2:  ;|CHECK|   ﾏｸﾛﾒｲ,|EM|
 3:  ;    |EM| ﾊ ｻﾌﾞﾙｰﾁﾝ ﾄﾋﾞｺｼ ﾊﾞﾝﾁ
 4:  ;          ｶﾅﾗｽﾞ  |LOCAL| ﾆｽﾙ
 5:  ;
 6:  CHECK     MACRO   MNAME,EM
 7:            IF      20H AND NOT TYPE MNAME
 8:  MNAME     EQU     $+3
 9:            ENDIF
10:            IF      MNAME EQ $+3
11:            JP      EM
12:            ENDM
13:
14:  ;
15:  ;ﾚｼﾞｽﾀ ﾚﾝｿﾞｸ ﾛｰﾄﾞ  |(HL+)
16:  ;|LDIRR|   ﾚｼﾞｽﾀｸﾞﾝ,ﾚｼﾞｽﾀ
```

```
17:  ;       ｱﾄﾉ ﾚｼﾞｽﾀ ﾊ ¦INC¦ ｼﾅｲ
18:  ;
19:  LDIRR    MACRO    REGS,REG
20:           IRPC     QQ,REGS
21:           LD       QQ,(HL)
22:           INC      HL
23:           ENDM
24:           IFNB     <REG>
25:           LD       REG,(HL)
26:           ENDIF
27:           ENDM
28:  ;
29:  ;ﾚｼﾞｽﾀ ﾚﾝｿﾞｸ ｽﾄｱ ¦(HL+)
30:  ;¦STIR¦ ﾚｼﾞｽﾀｸﾞﾝ,ﾚｼﾞｽﾀ
31:  ;
32:  ;
33:  STIR     MACRO    REGS,REG
34:           IRPC     QQ,REGS
35:           LD       (HL),QQ
36:           INC      HL
37:           ENDM
38:           IFNB     <REG>
39:           LD       (HL),REG
40:           ENDIF
41:           ENDM
42:  ;
43:  ;¦32BIT MUL,DIV MACRO
44:  ; ¦MULQ MP1,MP2,DST
45:  ;   ¦######.## * ##.###### -> ######.##
46:  ; ¦DIVQ DVDD,DVDR,DST
47:  ;   ¦######.## / ##.###### -> ######.##
48:  ; ｻｲ ｶｲ ｹﾀ ｼｼｬｺﾞﾆｭｳ｡ ｱﾏﾘ ﾌｶ
49:  ;
50:  MULQ     MACRO    MP1,MP2,DST
51:           LOCAL    EM,MQ1,MQ2,MQ3,MQ4
52:           LD       HL,MP1
53:           LD       IX,MP2
54:           CALL     MULQ
55:           IFNB     <DST>
56:           EX       DE,HL
57:           LD       HL,DST
58:           LD       A,(MPBF+3)
59:           STIR     AED,C      ;¦CDEA¦ﾆ ｺﾀｴｱﾘ
60:           ENDIF
61:           CHECK    MULQ,EM
62:           LD       IY,MPBF
63:           LD       DE,MPBF
64:           LD       BC,4
65:           LDIR
66:           LD       E,(IX)
67:           LD       D,(IX+1)
68:           LD       C,(IX+2)
69:           LD       B,(IX+3)
70:           LD       HL,0
71:           PUSH     HL
72:           LD       HL,0
```

```
 73:            LD      A,33       ;ｶｳﾝﾀ
 74:            SRL     (IY+3)
 75:  MQ1:      RR      (IY+2)
 76:            RR      (IY+1)
 77:            RR      (IY)
 78:            DEC     A
 79:            JR      Z,MQ4
 80:            JR      NC,MQ3
 81:            ADD     HL,DE
 82:            EX      (SP),HL
 83:            ADC     HL,BC
 84:  MQ2:      RR      H
 85:            RR      L
 86:            EX      (SP),HL
 87:            RR      H
 88:            RR      L
 89:            RR      (IY+3)
 90:            JR      MQ1
 91:  MQ3:      EX      (SP),HL
 92:            JR      MQ2
 93:  MQ4:      POP     BC         ;|BCHL| ﾆ ｺﾀｴ
 94:            LD      A,(MPBF+3)
 95:            BIT     7,(IY+3)  ;|4/5
 96:            RET     Z          ;|4|ｼｬ
 97:            INC     L          ;|5|ﾆｭｳ
 98:            RET     NZ
 99:            INC     H
100:            RET     NZ
101:            INC     C
102:            RET     NZ
103:            INC     B
104:            RET
105:  EM:
106:            ENDIF
107:            ENDM
108:
109:  DIVQ      MACRO   DVDD,DVDR,DST
110:            LOCAL   EM,DV1,DV2,DV3
111:            LD      HL,DVDD
112:            LD      IX,DVDR
113:            CALL    DIVQ
114:            IFNB    <DST>
115:            LD      (DST),HL ;|DEHL| ﾆ ｺﾀｴ
116:            LD      (DST+2),DE
117:            ENDIF
118:            CHECK   DIVQ,EM
119:            XOR     A
120:            LD      (MPBF+2),A
121:            LD      B,A
122:            LDIRR   AED,C
123:            PUSH    BC
124:            LD      HL,0
125:            LD      (MPBF),HL
126:            EX      DE,HL
127:            LD      E,(IX)
128:            LD      D,(IX+1)
```

```
129:            LD      C,(IX+2)
130:            LD      B,(IX+3)
131:            LD      IY,MPBF
132:            LD      A,34        ;|Acc| ｶｳﾝﾀ
133:  DV1:      AND     A
134:            SBC     HL,DE
135:            EX      (SP),HL
136:            SBC     HL,BC       ;|HL'
137:            DEC     A
138:            JR      Z,DV3
139:            JR      NC,DV2
140:            EX      (SP),HL
141:            ADD     HL,DE
142:            EX      (SP),HL
143:            ADC     HL,BC       ;|HL'
144:  DV2:      CCF
145:            EX      (SP),HL
146:            IRP     REG,<(IY),(IY+1),(IY+2),(IY+3),L,H>
147:            RL      REG
148:            ENDM
149:            EX      (SP),HL
150:            RL      L
151:            RL      H           ;|HL'
152:            EX      (SP),HL
153:            JR      DV1
154:  DV3:      POP     BC
155:            LD      DE,(MPBF+2)
156:            LD      HL,(MPBF)
157:            RET     C           ;|4|ｼｬ
158:            INC     L           ;|5|ﾆｭｳ
159:            RET     NZ
160:            INC     H
161:            RET     NZ
162:            INC     E
163:            RET     NZ
164:            INC     D
165:            RET
166:  EM:
167:            ENDIF
168:            ENDM
169:  ;
170:  ;|32BIT SQRT (INTGR,REAL) MACRO
171:  ;|SQRT  SRC,DST (INTGER)
172:  ;|     ######## --> ####
173:  ;|SQRT32         SRC,DST (REAL)
174:  ;|     ######.## --> ###.######
175:  ;
176:  SQRT      MACRO   SRC,DST
177:            LOCAL   EM,SQ1,SQ2,SQ3,SQ4,HDX2
178:            LD      HL,SRC
179:            CALL    SQRT
180:            IFNB    <DST>
181:            LD      HL,DST
182:            STIR    C,B
183:            ENDIF
184:            CHECK   SQRT,EM
```

```
185:            LDIRR   CBE,D
186:            PUSH    BC
187:            POP     IX          ;¦DEIX¦ ニ モトノ カズ゛
188:            LD      HL,0
189:            LD      BC,1
190:            LD      A,16        ;カウンタ
191:    HDX2    MACRO
192:            ADD     IX,IX
193:            RL      E
194:            RL      D
195:            ADC     HL,HL
196:            ADD     IX,IX
197:            RL      E
198:            RL      D
199:            ADC     HL,HL
200:            ENDM
201:    SQ1:    HDX2
202:            JR      NZ,SQ3
203:            DEC     A
204:            JR      NZ,SQ1
205:            DEC     C
206:            RET
207:    SQ2:    HDX2
208:            SCF
209:            RL      C
210:            RL      B
211:    SQ3:    SBC     HL,BC
212:            JR      NC,SQ4
213:            ADD     HL,BC
214:            RES     0,C
215:            JR      $+3
216:    SQ4:    INC     C
217:            DEC     A
218:            JR      NZ,SQ2
219:            SRL     B
220:            RR      C
221:            RET
222:    EM:
223:            ENDIF
224:            ENDM
225:    ;
226:    SQRT32  MACRO   SRC,DST
227:            LOCAL   EM,SQ1,SQ2,SQ3,SQ4
228:            LD      HL,SRC
229:            CALL    SQRT32
230:            IFNB    <DST>
231:            LD      HL,DST
232:            STIR    EDC,B
233:            ENDIF
234:            CHECK   SQRT32,EM
235:            LDIRR   EDC,B
236:            PUSH    DE
237:            POP     IX
238:            LD      (MPBF),BC
239:            LD      IY,MPBF
240:            LD      HL,0
```

```
241:            LD      C,H
242:            LD      B,L
243:            LD      DE,1
244:            PUSH    HL
245:            LD      A,32      ;ｶｳﾝﾀ
246:    SQ1:    REPT    2
247:            ADD     IX,IX
248:            RL      (IY)
249:            RL      (IY+1)
250:            ADC     HL,HL     ;|2BIT| ｼﾌﾄ ｦ
251:            ENDM
252:            JR      NZ,SQ3    ;ｾﾞﾛﾃﾞﾅｲ ｶ
253:            DEC     A
254:            JR      NZ,SQ1    ;ｵﾜﾙﾏﾃﾞ ｸﾘｶｴｽ
255:            DEC     E
256:            POP     HL
257:            RET
258:    SQ2:    REPT    2
259:            ADD     IX,IX
260:            RL      (IY)
261:            RL      (IY+1)
262:            ADC     HL,HL     ;|2BIT| ｼﾌﾄ ｦ
263:            EX      (SP),HL
264:            ADC     HL,HL
265:            EX      (SP),HL
266:            ENDM
267:            SCF
268:            RL      E
269:            RL      D
270:            RL      C
271:            RL      B
272:    SQ3:    SBC     HL,DE
273:            EX      (SP),HL
274:            SBC     HL,BC
275:            EX      (SP),HL
276:            INC     E
277:            JR      NC,SQ4
278:            ADD     HL,DE
279:            EX      (SP),HL
280:            ADC     HL,BC
281:            EX      (SP),HL
282:            RES     0,E
283:            JR      $+3
284:    SQ4:    INC     E
285:            DEC     A
286:            JR      NZ,SQ2
287:            SRL     B
288:            RR      C
289:            RR      D
290:            RR      E
291:            POP     HL
292:            RET               ;|BCDE| ﾆ ｺﾀｴ
293:    EM:
294:            ENDIF
295:            ENDM
296:    ;|
```

```
297:  ;ﾒﾓﾘ ｾﾂﾔｸ |BCD --> BINARY| ﾍﾝｶﾝ
298:  ;|DEBI  SRC,DST  ;8|ｹﾀ |BCD| ｦ |4|ﾊﾞｲﾄﾆ
299:  ;|DEBI12  SRC,DST  ;12|ｹﾀ |BCD| ｦ |6|ﾊﾞｲﾄﾆ
300:  ;|BIDE  SRC,DST  ;4|ﾊﾞｲﾄ ｦ |8|ｹﾀ |BCD|ﾆ
301:  ;|BIDE12  SRC,DST  ;6|ﾊﾞｲﾄ ｦ |12|ｹﾀ |BCD|ﾆ
302:  ;
303:  DEBI     MACRO   SRC,DST
304:           LOCAL   EM,DEB1,DEB2,DEB3
305:           LD      HL,SRC
306:           CALL    DEBI
307:           IFNB    <DST>
308:           EX      DE,HL
309:           LD      HL,DST
310:           STIR    CBE,D
311:           ENDIF
312:           CHECK   DEBI,EM
313:           LDIRR   EDC,B
314:           LD      IY,MPBF
315:           LD      (IY+2),32 ;ｶｳﾝﾀ
316:  DEB1:    SRL     B
317:           IRP     REG,<C,D,E,H,L,(IY+1),(IY)>
318:           RR      REG
319:           ENDM
320:           IRPC    REG,BCDE
321:           LD      A,REG
322:           CALL    DEB2
323:           LD      REG,A
324:           ENDM
325:           DEC     (IY+2)
326:           JR      NZ,DEB1
327:           LD      BC,(MPBF) ;|HLBC : BINARY
328:           RET
329:  DEB2:    BIT     3,A
330:           JR      Z,DEB3
331:           SUB     3
332:  DEB3:    BIT     7,A
333:           RET     Z
334:           SUB     48
335:           RET
336:  EM:
337:           ENDIF
338:           ENDM
339:  ;
340:  DEBI12   MACRO   SRC,DST
341:           LOCAL   EM,DEB1,DEB2,DEB3
342:           LD      HL,SRC
343:           CALL    DEBI12
344:           IFNB    <DST>
345:           LD      HL,MPBF
346:           LD      DE,DST
347:           LD      BC,6
348:           LDIR
349:           ENDIF
350:           CHECK   DEBI12,EM
351:           LDIRR   ED
352:           PUSH    DE
```

```
353:            LDIRR   EDC,B
354:            POP     HL     ;|BCDEHL| ニ モトノ カズ
355:            LD      IY,MPBF
356:            LD      (IY+6),48 ;カウンタ
357:  DEB1:     SRL     B
358:   IRP REG,<C,D,E,H,L,(IY+5),(IY+4),(IY+3),(IY+2),(IY+1),(IY)>
359:            RR      REG
360:            ENDM
361:            IRPC    REG,BCDEHL
362:            LD      A,REG
363:            CALL    DEB2
364:            LD      REG,A
365:            ENDM
366:            DEC     (IY+6)
367:            JR      NZ,DEB1
368:            LD      HL,(MPBF)  ;|BCDEHL : BINARY
369:            LD      DE,(MPBF+2)
370:            LD      BC,(MPBF+4)
371:            RET
372:  DEB2:     BIT     3,A
373:            JR      Z,DEB3
374:            SUB     3
375:  DEB3:     BIT     7,A
376:            RET     Z
377:            SUB     48
378:            RET
379:  EM:
380:            ENDIF
381:            ENDM
382:  ;
383:  BIDE      MACRO   SRC,DST
384:            LOCAL   EM,BID1
385:            LD      HL,SRC
386:            CALL    BIDE
387:            IFNB    <DST>
388:            LD      HL,DST
389:            STIR    EDC,B      ;|BCDE|ニ コタエアリ
390:            ENDIF
391:            CHECK   BIDE,EM
392:            LDIRR   CBE,D
393:            PUSH    BC
394:            POP     IX
395:            EX      DE,HL      ;|HLIX|ニ モトノカズ
396:            LD      BC,0
397:            LD      E,C
398:            LD      D,B
399:            LD      IY,MPBF
400:            LD      (IY),32 ;カウンタ
401:  BID1:     ADD     IX,IX
402:            ADC     HL,HL
403:            IRPC    REG,EDCB
404:            LD      A,REG
405:            ADC     A,A
406:            DAA
407:            LD      REG,A
408:            ENDM
```

```
409:            DEC     (IY)
410:            JR      NZ,BID1
411:            RET
412:  EM:
413:            ENDIF
414:            ENDM
415:  BIDE12    MACRO   SRC,DST
416:            LOCAL   EM,BID1
417:            LD      HL,SRC
418:            CALL    BIDE12
419:            IFNB    <DST>
420:            EX      DE,HL
421:            PUSH    HL
422:            LD      HL,DST
423:            STIR    ED          ;|BC(SP)DE|ニ コタエアリ
424:            POP     DE
425:            STIR    EDC,B
426:            ENDIF
427:            CHECK   BIDE12,EM
428:            LDIRR   CB
429:            PUSH    BC
430:            POP     IX          ;|IX| ニ シモ ノ ケタ
431:            LD      DE,MPBF
432:            LD      BC,4
433:            LDIR
434:            LD      BC,0
435:            LD      E,C
436:            LD      D,B
437:            LD      H,B
438:            LD      L,C
439:            LD      IY,MPBF
440:            LD      (IY+4),48 ;カウンタ
441:  BID1:     ADD     IX,IX
442:            IRPC    P,0123
443:            RL      (IY+P)
444:            ENDM
445:            IRPC    REG,LHEDCB
446:            LD      A,REG
447:            ADC     A,A
448:            DAA
449:            LD      REG,A
450:            ENDM
451:            DEC     (IY+4)
452:            JR      NZ,BID1
453:            RET
454:  EM:
455:            ENDIF
456:            ENDM
457:  ;
458:  ;|N|バイト |--> ASCII 16|シン ヘンカン
459:  ;|HXASC SRC,DST,N ;N| シテイ ナシ ハ |4
460:  ;|      DST| ハ |2*N|バイト エリヤ
461:  ;
462:  HXASC     MACRO   SRC,DST,N
463:            LOCAL   EM
464:            IFB     <N>
```

```
465:                LD      B,4
466:                LD      HL,DST+7
467:                ELSE
468:                LD      B,N
469:                LD      HL,DST+2*N-1
470:                ENDIF
471:                LD      DE,SRC
472:                CALL    HXASC
473:                CHECK   HXASC,EM
474:                LD      A,(DE)'
475:                OR      240
476:                DAA
477:                ADD     A,160
478:                ADC     A,64
479:                LD      (HL),A
480:                LD      A,(DE)
481:                DEC     HL
482:                RRA
483:                RRA
484:                RRA
485:                RRA
486:                OR      240
487:                DAA
488:                ADD     A,160
489:                ADC     A,64
490:                LD      (HL),A
491:                DEC     HL
492:                INC     DE
493:                DJNZ    HXASC
494:                RET
495:    EM:
496:                ENDIF
497:                ENDM
498:    ;
499:    ;¦ASCII 16¦ｼﾝ ¦--> N¦ﾊﾞｲﾄ ﾍﾝｶﾝ
500:    ;¦ASCHX SRC,DST,N   ;N¦ ｼﾃｲﾅｼ ﾊ ¦4
501:    ;¦        SRC¦ ﾊ ¦2*N¦ ﾓｼﾞ｡ ¦$,  ,cr,lf,tab,.
502:    ;¦                                       ¦ｶﾞ ｱﾚﾊﾞ  ｳﾁｷﾘ
503:    ;
504:    ASCHX       MACRO   SRC,DST,N
505:                LOCAL   EM,AS1,AS2,AS3
506:                IFB     <N>
507:                LD      B,4
508:                LD      HL,DST+3
509:                ELSE
510:                LD      B,N
511:                LD      HL,DST+N-1
512:                ENDIF
513:                LD      DE,SRC
514:                CALL    ASCHX
515:                CHECK   ASCHX,EM
516:                PUSH    BC
517:                XOR     A
518:                LD      (HL),A
519:                DEC     HL
520:                DJNZ    $-2
```

```
521:            INC     HL
522:            PUSH    HL          ;セントウ バンチ
523:            DEC     DE
524:            INC     DE
525:            LD      A,(DE)
526:            CP      21H
527:            JR      C,$-4  ;ミエナイモジ ヨミトバシ
528:    AS1:    CP      '0'
529:            JR      C,AS3
530:            CP      ':'
531:            JR      C,AS2
532:            SUB     7
533:    AS2:    POP     HL  ;ウエノ ニブル ムシ
534:            POP     BC
535:            PUSH    BC
536:            PUSH    HL
537:            RLD
538:            INC     HL
539:            DJNZ    $-3
540:            INC     DE
541:            LD      A,(DE)
542:            JR      AS1
543:    AS3:    POP     HL
544:            POP     BC
545:            RET
546:    EM:
547:            ENDIF
548:            ENDM
549:    ;
```

〈図 5.19〉 高級言語風 4 バイト四則インタプリタ

(a) ソース・リスト

```
  1:            .Z80
  2:    ; コーキューゲンゴ フー マクロ
  3:    ; |4|バイト エンザン
  4:    ;
  5:            .XLIST
  6:    ; シキ ノ カイセキ
  7:    ;
  8:    LET     MACRO   P1,DD,P2,P3,P4 ;DD| ハ ダミー
  9:            IFIDN   <P3>,<+>
 10:            DW      ADDQ
 11:            ENDIF
 12:            IFIDN   <P3>,<->
 13:            DW      SUBQ
 14:            ENDIF
 15:            IFIDN   <P3>,<*>
 16:            DW      MULQ
 17:            ENDIF
 18:            IFIDN   <P3>,</>
 19:            DW      DIVQ
 20:            ENDIF
 21:            DW      P2,P4,P1
```

```
22:             ENDM
23:     ;
24:     ;ｼﾞｬﾑﾌﾟ ﾒｲﾚｲ
25:     ;
26:     GOTO    MACRO   ADS
27:             LD      DE,ADS
28:             RET
29:             ENDM
30:     ;
31:     ;ｶﾝｾﾂ ﾛｰﾄﾞ ｵｰﾄ ｲﾝｸﾘ ﾏｸﾛ
32:     ;
33:     LDP     MACRO   DST,SRC
34:             IRPC    X,DST
35:             LD      A,(SRC)
36:             LD      X,A
37:             INC     SRC
38:             ENDM
39:             ENDM
40:     ;
41:     PUPO    MACRO   SRC,DST
42:             PUSH    SRC
43:             POP     DST
44:             ENDM
45:             .LIST
46:     ;
47:     ;ｲﾝﾀﾌﾟﾘﾀ ﾒｲﾝ
48:     ;
49:     START:  LD      DE,STCODE   ;ｲﾝﾀﾌﾟﾘﾀ ｺｰﾄﾞ ﾉ ｽﾀｰﾄ
50:             LD      HL,$   ;ｼｮﾘ ｶﾗﾉ ﾘﾀﾝ ｱﾄﾞﾚｽ ｦ ｽﾀｯｸﾍ
51:             PUSH    HL
52:             LDP     LH,DE     ;|HL| ﾆ ｺﾍﾞﾂ ｼｮﾘ ｱﾄﾞﾚｽ
53:             JP      (HL)
54:     ;
55:     ;ｺﾍﾞﾂ ｴﾝｻﾞﾝ ｼｮﾘ
56:     ;
57:     ADDQ:   LDP     CBLH,DE
58:             PUPO    HL,IX
59:             LDP     LH,DE
60:             PUSH    DE
61:             PUPO    HL,IY
62:             LDP     LH,BC
63:             LD      E,(IX)
64:             LD      D,(IX+1)
65:             ADD     HL,DE
66:             LD      (IY),L
67:             LD      (IY+1),H
68:             LDP     LH,BC
69:             LD      E,(IX+2)
70:             LD      D,(IX+3)
71:             ADC     HL,DE
72:             LD      (IY+2),L
73:             LD      (IY+3),H
74:             POP     DE
75:             RET
76:     ;
77:     SUBQ:   LDP     CBLH,DE
```

```
 78:           PUPO    HL,IX
 79:           LDP     LH,DE
 80:           PUSH    DE
 81:           PUPO    HL,IY
 82:           LDP     LH,BC
 83:           LD      E,(IX)
 84:           LD      D,(IX+1)
 85:           AND     A
 86:           SBC     HL,DE
 87:           LD      (IY),L
 88:           LD      (IY+1),H
 89:           LDP     LH,BC
 90:           LD      E,(IX+2)
 91:           LD      D,(IX+3)
 92:           SBC     HL,DE
 93:           LD      (IY+2),L
 94:           LD      (IY+3),H
 95:           POP     DE
 96:           RET
 97:   ;
 98:   MULQ:
 99:                   ;ｼｮｳｻｲ ﾊ ｼｮｳﾘｬｸ
100:           REPT    6
101:           INC     DE
102:           ENDM
103:           RET
104:   ;
105:   DIVQ:
106:                   ;ｼｮｳｻｲ ﾊ ｼｮｳﾘｬｸ
107:           REPT    6
108:           INC     DE
109:           ENDM
110:           RET
111:   ;
112:   ;
113:   AA:     DW      5678H,1234H
114:   BB:     DW      3344H,1122H
115:   CC:     DS      4
116:   ;
117:   ; ｺｰｷｭｰｹﾞﾝｺﾞ ﾌｰ ﾏｸﾛ ﾖﾋﾞﾀﾞｼ
118:   ;
119:   ; ｽﾍﾞﾃ |4|ﾊﾞｲﾄ|(32|ﾋﾞｯﾄ|)| ﾉ ｴﾝｻﾞﾝ
120:   ;
121:   STCODE: LET     CC := AA + BB
122:           LET     CC := AA - BB
123:           LET     CC := AA * BB
124:           LET     CC := AA / BB
125:   ;
126:           GOTO    STCODE  ;ｴﾝﾄﾞﾚｽ ﾙｰﾌﾟ
127:   ;
128:           END     START
```

(b) アセンブル・リスト

```
                                        .Z80
                                ; ｺｰｷｭｰｹﾞﾝｺﾞ ﾌｰ ﾏｸﾛ
                                ; 4ﾊﾞｲﾄ ｴﾝｻﾞﾝ
                                ;
                                        .LIST
                                ;
                                ;ｲﾝﾀﾌﾟﾘﾀ ﾒｲﾝ
                                ;
0000'   11 00AE'                START:  LD      DE,STCODE  ;ｲﾝﾀﾌﾟﾘﾀ ｺｰﾄﾞ ﾉ ｽﾀｰﾄ
0003'   21 0003'                        LD      HL,$  ;ｼｮﾘ ｶﾗﾉ ﾘﾀﾝ ｱﾄﾞﾚｽ ｦ ｽﾀｯｸﾍ
0006'   E5                              PUSH    HL
                                        LDP     LH,DE   ;HL ﾆ ｺﾍﾞﾂ ｼｮﾘ ｱﾄﾞﾚｽ
0007'   1A              +               LD      A,(DE)
0008'   6F              +               LD      L,A
0009'   13              +               INC     DE
000A'   1A              +               LD      A,(DE)
000B'   67              +               LD      H,A
000C'   13              +               INC     DE
000D'   E9                              JP      (HL)
                                ;
                                ;ｺﾍﾞﾂ ｴﾝｻﾞﾝ ｼｮﾘ
                                ;
000E'                           ADDQ:   LDP     CBLH,DE
000E'   1A              +               LD      A,(DE)
000F'   4F              +               LD      C,A
0010'   13              +               INC     DE
0011'   1A              +               LD      A,(DE)
0012'   47              +               LD      B,A
0013'   13              +               INC     DE
0014'   1A              +               LD      A,(DE)
0015'   6F              +               LD      L,A
0016'   13              +               INC     DE
0017'   1A              +               LD      A,(DE)
0018'   67              +               LD      H,A
0019'   13              +               INC     DE
                                        PUPO    HL,IX
001A'   E5              +               PUSH    HL
001B'   DD E1           +               POP     IX
                                        LDP     LH,DE
001D'   1A              +               LD      A,(DE)
001E'   6F              +               LD      L,A
001F'   13              +               INC     DE
0020'   1A              +               LD      A,(DE)
0021'   67              +               LD      H,A
0022'   13              +               INC     DE
0023'   D5                              PUSH    DE
                                        PUPO    HL,IY
0024'   E5              +               PUSH    HL
0025'   FD E1           +               POP     IY
                                        LDP     LH,BC
0027'   0A              +               LD      A,(BC)
0028'   6F              +               LD      L,A
0029'   03              +               INC     BC
002A'   0A              +               LD      A,(BC)
```

```
002B'   67              +           LD      H,A
002C'   03              +           INC     BC
002D'   DD 5E 00                    LD      E,(IX)
0030'   DD 56 01                    LD      D,(IX+1)
0033'   19                          ADD     HL,DE
0034'   FD 75 00                    LD      (IY),L
0037'   FD 74 01                    LD      (IY+1),H
                                    LDP     LH,BC
003A'   0A              +           LD      A,(BC)
003B'   6F              +           LD      L,A
003C'   03              +           INC     BC
003D'   0A              +           LD      A,(BC)
003E'   67              +           LD      H,A
003F'   03              +           INC     BC
0040'   DD 5E 02                    LD      E,(IX+2)
0043'   DD 56 03                    LD      D,(IX+3)
0046'   ED 5A                       ADC     HL,DE
0048'   FD 75 02                    LD      (IY+2),L
004B'   FD 74 03                    LD      (IY+3),H
004E'   D1                          POP     DE
004F'   C9                          RET
                        ;
0050'                   SUBQ:       LDP     CBLH,DE
0050'   1A              +           LD      A,(DE)
0051'   4F              +           LD      C,A
0052'   13              +           INC     DE
0053'   1A              +           LD      A,(DE)
0054'   47              +           LD      B,A
0055'   13              +           INC     DE
0056'   1A              +           LD      A,(DE)
0057'   6F              +           LD      L,A
0058'   13              +           INC     DE
0059'   1A              +           LD      A,(DE)
005A'   67              +           LD      H,A
005B'   13              +           INC     DE
                                    PUPO    HL,IX
005C'   E5              +           PUSH    HL
005D'   DD E1           +           POP     IX
                                    LDP     LH,DE
005F'   1A              +           LD      A,(DE)
0060'   6F              +           LD      L,A
0061'   13              +           INC     DE
0062'   1A              +           LD      A,(DE)
0063'   67              +           LD      H,A
0064'   13              +           INC     DE
0065'   D5                          PUSH    DE
                                    PUPO    HL,IY
0066'   E5              +           PUSH    HL
0067'   FD E1           +           POP     IY
                                    LDP     LH,BC
0069'   0A              +           LD      A,(BC)
006A'   6F              +           LD      L,A
006B'   03              +           INC     BC
006C'   0A              +           LD      A,(BC)
006D'   67              +           LD      H,A
006E'   03              +           INC     BC
```

```
006F'   DD 5E 00            LD      E,(IX)
0072'   DD 56 01            LD      D,(IX+1)
0075'   A7                  AND     A
0076'   ED 52               SBC     HL,DE
0078'   FD 75 00            LD      (IY),L
007B'   FD 74 01            LD      (IY+1),H
                            LDP     LH,BC
007E'   0A           +      LD      A,(BC)
007F'   6F           +      LD      L,A
0080'   03           +      INC     BC
0081'   0A           +      LD      A,(BC)
0082'   67           +      LD      H,A
0083'   03           +      INC     BC
0084'   DD 5E 02            LD      E,(IX+2)
0087'   DD 56 03            LD      D,(IX+3)
008A'   ED 52               SBC     HL,DE
008C'   FD 75 02            LD      (IY+2),L
008F'   FD 74 03            LD      (IY+3),H
0092'   D1                  POP     DE
0093'   C9                  RET
                    ;
0094'               MULQ:
                                    ;ｼｮｳｻｲ ﾊ ｼｮｳﾘｬｸ
                            REPT    6
                            INC     DE
                            ENDM
0094'   13           +      INC     DE
0095'   13           +      INC     DE
0096'   13           +      INC     DE
0097'   13           +      INC     DE
0098'   13           +      INC     DE
0099'   13           +      INC     DE
009A'   C9                  RET
                    ;
009B'               DIVQ:
                                    ;ｼｮｳｻｲ ﾊ ｼｮｳﾘｬｸ
                            REPT    6
                            INC     DE
                            ENDM
009B'   13           +      INC     DE
009C'   13           +      INC     DE
009D'   13           +      INC     DE
009E'   13           +      INC     DE
009F'   13           +      INC     DE
00A0'   13           +      INC     DE
00A1'   C9                  RET
                    ;
                    ;
00A2'   5678 1234   AA:     DW      5678H,1234H
00A6'   3344 1122   BB:     DW      3344H,1122H
00AA'               CC:     DS      4
                    ;
                    ; ｺｰｷｭｰｹﾞﾝｺﾞ ﾌｰ ﾏｸﾛ ﾖﾋﾞﾀﾞｼ
                    ;
                    ; ｽﾍﾞﾃ 4ﾊﾞｲﾄ(32ﾋﾞｯﾄ) ﾉ ｴﾝｻﾞﾝ
                    ;
00AE'               STCODE: LET     CC := AA + BB
```

```
00AE'   000E'         +      DW     ADDQ
00B0'   00A2' 00A6'   +      DW     AA,BB,CC
00B4'   00AA'         +
                             LET    CC := AA - BB
00B6'   0050'         +      DW     SUBQ
00B8'   00A2' 00A6'   +      DW     AA,BB,CC
00BC'   00AA'         +
                             LET    CC := AA * BB
00BE'   0094'         +      DW     MULQ
00C0'   00A2' 00A6'   +      DW     AA,BB,CC
00C4'   00AA'         +
                             LET    CC := AA / BB
00C6'   009B'         +      DW     DIVQ
00C8'   00A2' 00A6'   +      DW     AA,BB,CC
00CC'   00AA'         +
                        ;
                             GOTO   STCODE  ;ｴﾝﾄﾞﾚｽ ﾙｰﾌﾟ
00CE'   11 00AE'      +      LD     DE,STCODE
00D1'   C9            +      RET
                        ;
                             END    START
```

5.5 BPU用クロス・アセンブラ

ごく簡単な命令体系のプロセッサでも，新たにアセンブラを1本開発するとなると相当な手間がかかります．まして，普段アセンブラを利用することはあっても作る機会がない者にとってはまとまった大仕事といえるでしょう．

そこで，マクロ・アセンブラのマクロ機能を使用して，本来持っている機械語コードの機能を全く使わず，別のプロセッサ用のコードを生成するクロス・アセンブラ的に利用する方法を考えます．このためには，マクロ・アセンブラのすべての疑似命令が使えますから，アセンブラを新たに開発するよりははるかに少ない作業量で済みます．

ここでは別のプロセッサとして，ディジタル回路設計ノウハウ〔参考文献(2)〕の3.3節に紹介したビット・プロセッサ(BPU)を対象とし，その機械語コードを生成するためのマクロ定義をM80でアセンブルしますが，クロス・アセンブラはアセンブラ内蔵の機械語コードを使用しませんから，ほとんどのマクロ・アセンブラでも同様の方法でクロス・アセンブルできます．

BPUの詳細はここでは触れませんが，命令体系は簡単で図5.20に示す3種類です．BPUの命令コードを合成するためには**DB**と演算機能を利用します．たとえば，ジャンプ命令は，

〈図5.20〉 ビット・プロセッサ (BPU) 命令体系

種別	ビット・フォーマット アセンブラ表記	F フラグ	動作の内容 パラメータ	ハード的動作
出力命令	7 6 5 4 3 2 1 0 0 0 D A A A A A OUT n, P	変化しない	出力ポートnにPを出力するフラグがセットされていると実行しない (NOPとなる) n：0～31 P：0/1	F=0のとき(n)←P F=1のとき何もしない
入力命令	7 6 5 4 3 2 1 0 0 1 D A A A A A IN n, P	結果により決まる	入力ポートnより入るデータとPを比較し，一致しなければフラグをセットする．この命令はフラグをリセットすることはない n：0～31 P：0/1	(n)=Pのとき何もしない (n)=$\bar{P}$のときF←1
飛越命令	7 6 5 4 3 2 1 0 1 A A A A A A A JMP m	命令終了後リセット	m番地へジャンプする フラグがセットされているとNOPとなる．この命令はNOPになってもジャンプしてもフラグをリセットする m：0～127	F=0のときPC←m F=1のときF←0

D：データ A：アドレス F：フラグ

```
DB      ADRS  OR  80H
```

として計算されます．出力命令は**H**を**32**に，**L**を**0**に**EQU**しておけば，

```
DB      PORT  OR  P
```

となります．

マクロ命令には自由に名前が付けられますから，**JMP**を**GOTO**に，**IN**を**WHEN**に書き換えても定義さえすれば呼び出せます．このように，少し高級言語風にマクロ定義したクロス・アセンブラが図5.21です．出力命令は**OUT**でなく**SET**，**ON**，**RES**，**OFF**の4種類，**IN**は**IF**も使えるようにしました．このようなマクロ名なら，その意図は明白で説明を要しないと思います．

ところで，このようなクロス・アセンブラのリストは図5.21のようにすべて**DB** △△△という形式で，BPUのアセンブラ命令(実際はM80にとってマクロ命令だが，BPUとして見ればBPUの機械語命令)の行にはバイナリ・コードが出力されません．ソース・プログラムの上ではBPUの命令のみが並んでいて見やすいのに，リスト上では**DB**にじゃまされて見づらくなっています．

だからといって，マクロに展開をリストしない(**.SALL**)ように指示するとバイナリ・コードがリスト上から消えてしまいます．原理を理解するだけならどんなリストでも構いま

〈図5.21〉クロス・アセンブラのリスト

(a) ソース・リスト

```
 1:  INIT    MACRO
 2:          DEFB   $+1 OR 128
 3:          ENDM
 4:  WHEN    MACRO  A,B
 5:          IFIDN  <B>,<H>
 6:          DEFB   96+A
 7:          ELSE
 8:          DEFB   64+A
 9:          ENDIF
10:          ENDM
11:  IF      MACRO  A,B ;EVEN IF { IF } CAN BE A MACRO!!
12:          IFIDN  <B>,<H>
13:          DEFB   96+A
14:          ELSE
15:          DEFB   64+A
16:          ENDIF
17:          ENDM
18:  GOTO    MACRO  A
19:          DEFB   A OR 128
20:          ENDM
21:  SET     MACRO  A
22:          DEFB   A+32
23:          ENDM
24:  ON      MACRO  A
25:          DEFB   A+32
26:          ENDM
27:  RES     MACRO  A
28:          DEFB   A
29:          ENDM
30:  OFF     MACRO  A
31:          DEFB   A
32:          ENDM
33:  ;
34:  SWITCH  EQU    3
35:  RELAY   EQU    5
36:  NOB     EQU    7
37:  MOTOR   EQU    9
38:  LAMP    EQU    12
39:  ;
40:          ASEG
41:          INIT
42:  START:  WHEN   SWITCH,H
43:          SET    RELAY
44:          GOTO   START
45:          ;ENDWH
46:          RES    RELAY
47:          IF     NOB,H
48:          ON     MOTOR
49:          ON     LAMP
50:          ;
51:          OFF    MOTOR
52:          OFF    LAMP
53:          ;
54:          GOTO   START
55:          GOTO   START
56:          END
```

(41–55: メイン・プログラム すべてがマクロ命令)

(b) アセンブル・リスト

```
                              INIT     MACRO
                                       DEFB     $+1 OR 128
                                       ENDM
                              WHEN     MACRO    A,B
                                       IFIDN    <B>,<H>
                                       DEFB     96+A
                                       ELSE
                                       DEFB     64+A
                                       ENDIF
                                       ENDM
                              IF       MACRO    A,B ;EVEN IF ( IF ) CAN BE A MACRO!!
                                       IFIDN    <B>,<H>
                                       DEFB     96+A
                                       ELSE
                                       DEFB     64+A
                                       ENDIF
                                       ENDM
                              GOTO     MACRO    A
                                       DEFB     A OR 128
                                       ENDM
                              SET      MACRO    A
                                       DEFB     A+32
                                       ENDM
                              ON       MACRO    A
                                       DEFB     A+32
                                       ENDM
                              RES      MACRO    A
                                       DEFB     A
                                       ENDM
                              OFF      MACRO    A
                                       DEFB     A
                                       ENDM
                              ;
0003                          SWITCH   EQU      3
0005                          RELAY    EQU      5
0007                          NOB      EQU      7
0009                          MOTOR    EQU      9
000C                          LAMP     EQU      12
                              ;
0000                                   ASEG
                                       INIT
0000    81          +                  DEFB     $+1 OR 128
0001                          START:   WHEN     SWITCH,H
0001    63          +                  DEFB     96+SWITCH
                                       SET      RELAY
0002    25          +                  DEFB     RELAY+32
                                       GOTO     START
0003    81          +                  DEFB     START OR 128
                                       ;ENDWH
                                       RES      RELAY
0004    05          +                  DEFB     RELAY
                                       IF       NOB,H
0005    67          +                  DEFB     96+NOB
                                       ON       MOTOR
0006    29          +                  DEFB     MOTOR+32
                                       ON       LAMP
```

```
0007    2C          +       DEFB    LAMP+32
                            ;
                            OFF     MOTOR
0008    09          +       DEFB    MOTOR
                            OFF     LAMP
0009    0C          +       DEFB    LAMP
                            ;
                            GOTO    START
000A    81          +       DEFB    START OR 128
                            GOTO    START
000B    81          +       DEFB    START OR 128
                            END
```

〈図5.22〉見易い書式に変換されたリスト

```
                                ;
0003                            SWITCH  EQU     3
0005                            RELAY   EQU     5
0007                            NOB     EQU     7
0009                            MOTOR   EQU     9
000C                            LAMP    EQU     12
                                ;
0000                                    ASEG
0000    81          ¥                   INIT
0001    63          ¥           START:  WHEN    SWITCH,H
0002    25          ¥                   SET     RELAY
0003    81          ¥                   GOTO    START
                                        ;ENDWH
0004    05          ¥                   RES     RELAY
0005    67          ¥                   IF      NOB,H
0006    29          ¥                   ON      MOTOR
0007    2C          ¥                   ON      LAMP
                                        ;
0008    09          ¥                   OFF     MOTOR
0009    0C          ¥                   OFF     LAMP
                                        ;
000A    81          ¥                   GOTO    START
000B    81          ¥                   GOTO    START
                                        END
```

せんが，BPUを実現してそのアセンブラを常用するとき，この見づらさは作業性を極端に悪くします．そこで，リスト上から各**DB**の行を消して，なおかつバイナリ・コードを残す方法はないものかと考えましたが，アセンブラ内では簡単にできそうもないので，一度展開されたリスト・ファイルを作って，そのファイルからプリントするべき変換をして別のファイルにするリスト変換プログラムを作りました．

通常はマクロ展開行は＋の記号で示されていますが，変換プログラムを通ったものには

¥記号を付けました(図5.22). これでM80から直接得られたリストでないことがはっきりわかります. この記号を取り去ってしまうと, 一見BPU専用のアセンブラのように見えます.

BPU用に限らず, マクロ定義をクロス・アセンブラ用に使用する時は, このリスト変換は有用です. この変換プログラムもカナ変換と同じくPascal/MT+を使用して作りました.

5.6 ロジック・シミュレーター――論理式をマクロで解釈

制御用のシステムでは外部にリレーがつながったり, リレー・シーケンスに代わるプログラム・シーケンサがつながったりします. 外部のシーケンサもある程度以上の機能が必要ならば存在価値がありますが, 中にはリレー数個くらいの簡単なものもあり, このような場合は制御用コンピュータ内で処理できそうな気もします. システム購入者の側からも, この程度のものならコンピュータに組み込めないものかと相談されることもあるでしょう.

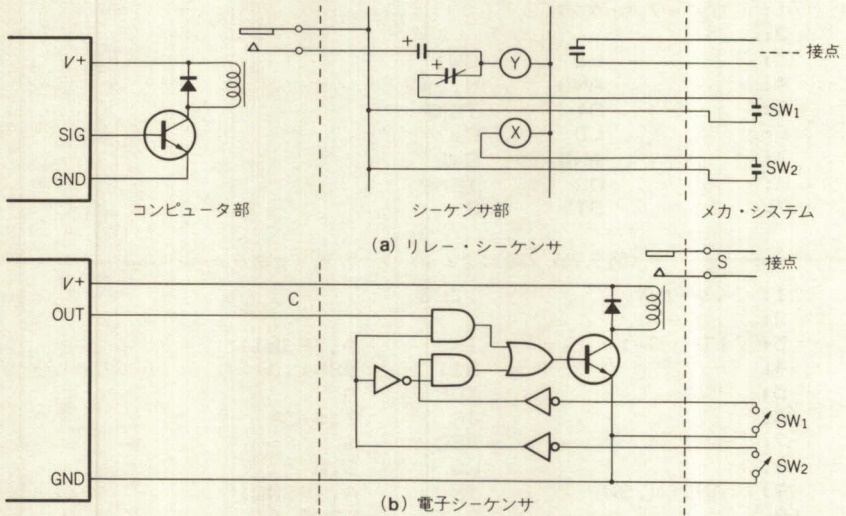
ところが, コンピュータ・システムの立場からすると, これが案外大変なことなのです. その理由は, シーケンサやリレー・シーケンスと呼ばれるものはコンピュータのようにフロー式の設計になっていないからです. たとえば, 図5.23のようなシーケンサを考えた場合, SW2を閉じなければ接点はSW1のON/OFFと同様にON/OFFします. コンピュータの出力にかかわらず常にこの動作は可能です. 同図(a)と(b)はリレー動作の時間おくれに差が出る以外は全く同じです. コンピュータにとって扱いにくいのは, この常に動作するというポイントです. コンピュータでシーケンサの動作だけをするのならこれは何でもありませんが, 他に主とした動作を行っていながら, 外部信号の動作を常に続けるというのはインタラプトによるしかありません.

では, 図5.23(b)のような回路をコンピュータのインターフェースに組み込んだらどうでしょうか. ICを2個追加するだけでできる回路ですから一見問題なさそうですが, 故障した時には他の回路(通常はプリント基板単位で保守用の部品を用意しておく)と共通性がないので即交換するわけには行きません. その1件だけならともかく, 毎回ちょっとした回路を追加するようなことをやっていたのではすべての部品に共通性がなくなり, メーカとしても故障対策が不十分な状態になってしまいます. ひどい場合には, 取り替えようとしてよく見たら基板についているICの数が違っていたという苦情を受ける結果になります.

そこでハード的な変更ではなく, 純粋にソフトウェアのみでこのようなシーケンサ部分

〈図 5.23〉 シーケンサ外付けの例

(a) リレー・シーケンサ

(b) 電子シーケンサ

を吸収する方法が必要になります．もちろん，そのためにはこのルーチンを含む部分をタイマのインタラプトで起動して常にメイン・ルーチンと無関係に動作させておく必要があります．

つぎに，このシーケンス回路やロジック回路をどのようにしてコーディングすればよいでしょうか．当然，回路図をそのままコーディングに使うことはできませんから，プログラム・シーケンサの書式で考えてみます．一般的なプログラムは図 5.24 のようになります．シーケンサの場合は1ビット単位のデータを扱うように設計されていますから，ロジックの動きは簡単に書けるのです．

これを Z80 で肩代わりしようとすると図 5.25 のように大きなものになり，実行速度も遅いものになります．Z80 はビット・データを扱う命令がないので条件判断と **BIT** 命令を組み合わせて使用していますが，それでもシーケンサのまねをしたのでは分が悪いようです．この書式のままでマクロ定義をしてシーケンサ用マクロとする方法もとれますが，もう一工夫欲しいところです．

図 5.25 では，シーケンサ風にロジック・レベルを C レジスタに置いて **AND** や **OR** を実行しています．これをデータとしてではなくフローの上で処理できないでしょうか．もと

〈図5.24〉 シーケンサのプログラムの例

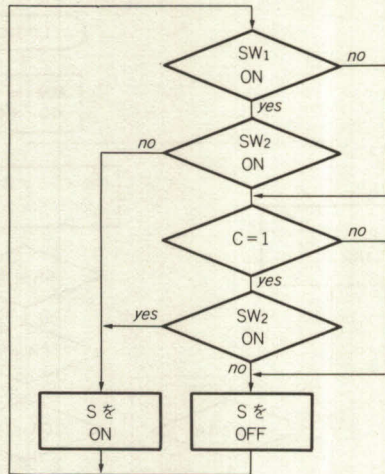
```
 1:  ﾛｼﾞｯｸ ｼｰｹﾝｻ
 2:
 3:          LD      SW1
 4:          AND     N,SW2
 5:          ST      TEMP
 6:          LD      C
 7:          AND     SW2
 8:          OR      TEMP
 9:          ST      S
```

〈図5.25〉 Z80によるシーケンサ風プログラム

```
 1:  .ｼｰｹﾝｻ             .Z80
 2:
 3:  LD   SW1           IN      A,(PSW1)
 4:                     BIT     BSW1,A
 5:                     XOR     A
 6:                     JR      Z,$+3
 7:                     DEC     A
 8:                     LD      C,A
 9:  AND N,SW2          IN      A,(PSW2)
10:                     BIT     BSW2,A
11:                     XOR     A
12:                     JR      NZ,$+3
13:                     DEC     A
14:                     AND     C
15:  ST   TEMP          LD      (TEMP),A
16:  LD   C             IN      A,(PC)
17:                     BIT     BC,A
18:                     XOR     A
19:                     JR      Z,$+3
20:                     DEC     A
21:                     LD      C,A
22:  AND SW2            IN      A,(PSW2)
23:                     BIT     BSW2,A
24:                     XOR     A
25:                     JR      Z,$+3
26:                     DEC     A
27:                     AND     C
28:  OR   TEMP          LD      C,A
29:                     LD      A,(TEMP)
30:                     OR      C
31:                     LD      C,A
32:  ST   S             IN      A,(PS)
33:                     BIT     0,C
34:                     SET     BS,A
35:                     JR      NZ,$+4
36:                     RES     BS,A
37:                     OUT     (PS),A
```

〈図5.26〉 図5.23の解析

$SW_1 \cdot \overline{SW_2} + C \cdot SW_2 \rightarrow S$

(a) 論理式

(b) フローチャート

もとコンピュータには，用もないのにグルグルと入力信号の **AND** や **OR** を取って回っているというのは向いていません．もっと必要最小限の動作の方が適しています．

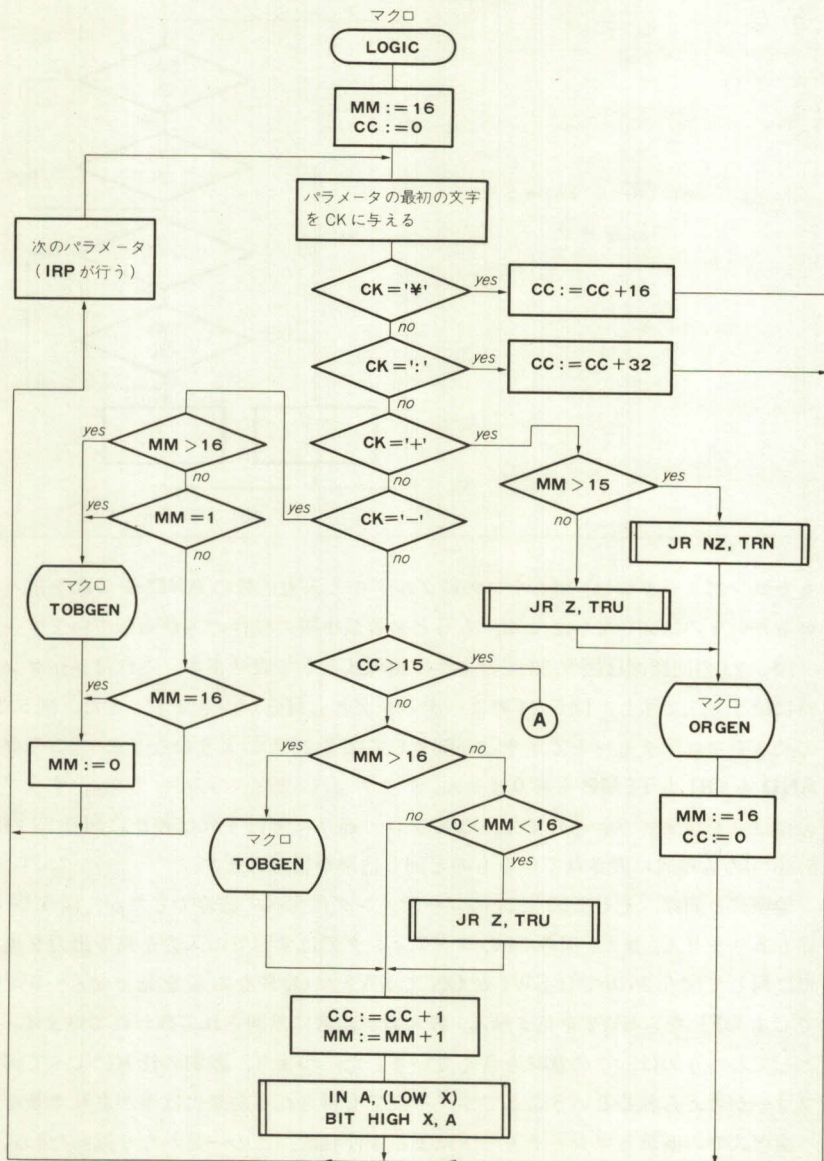
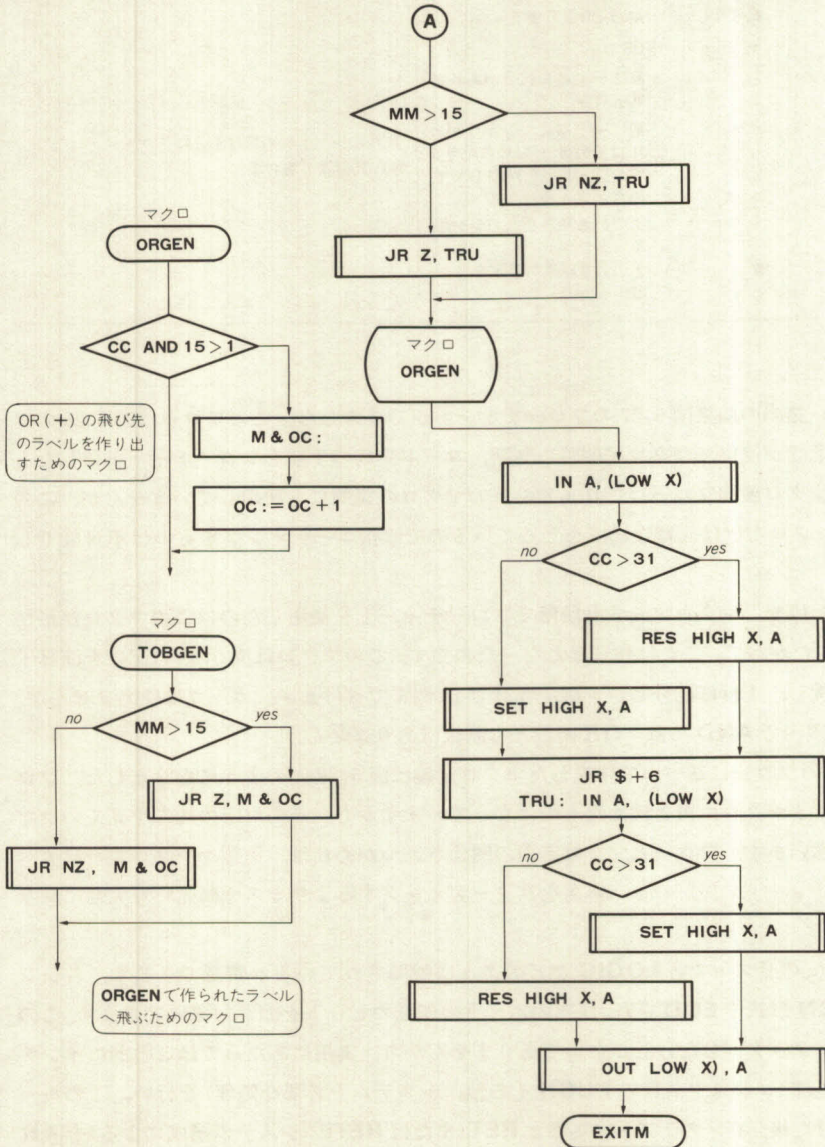
図5.23の回路図は図5.26(a)のように論理式として表せます．これはシーケンスを組む時に使う方法ですし，図5.24のコーディングとも対応がつきます．また，図5.23の回路の動きをフローチャートで示すと一例として図5.26(b)のようになります．このフローには **AND** も **OR** も **TEMP** もありません．データという概念が存在しません．すべてはフローの中に盛り込まれているのです．このフロー通りに実行すれば確かに図5.23の回路や図5.26(a)の論理式に表されているものと同じ結果が得られます．

論理式と回路，そして図5.24のコーディングはすべて演算でしたが，図5.26(b)には演算もありません．また，図5.24のコーディングではすべての入力を見て出力を出しているのに対して図5.26(b)ではSW1がONでSW2がOFFならCを見ません．SW1がOFFでCもOFFならSW2を見ません．各入力が非常に差別されて扱われています．先の一例としてというのは，この意味を含んでいました．つまり，差別の仕方によって何種類ものフローが考えられるということです．それでも得られる結果には差がありません．

論理式での演算とフローチャートによる条件判断．この一見かなり違った形式で，しかも前述のように本質的に異なる原理のものを結びつけるマクロ定義ができないものかと試

〈図5.27〉 マクロLOGICの

展開フローチャート

A
MM > 15
JR NZ, TRU
JR Z, TRU
マクロ
ORGEN
マクロ
ORGEN
CC AND 15 > 1
OR (+) の飛び先のラベルを作り出すためのマクロ
M & OC :
OC := OC + 1
IN A, (LOW X)
no
CC > 31
yes
RES HIGH X, A
SET HIGH X, A
マクロ
TOBGEN
no
MM > 15
yes
JR Z, M & OC
JR NZ, M & OC
JR $ + 6
TRU : IN A, (LOW X)
no
CC > 31
yes
SET HIGH X, A
RES HIGH X, A
OUT LOW X) , A
EXITM
ORGENで作られたラベルへ飛ぶためのマクロ

〈図 5.28〉 マクロ LOGIC が解釈できる演算子と記号

記号	意味	
記号なし	AND, ORより優先	記号とシンボルの数は1行に合計16個まで書ける
+	OR	
−	前に −，後に記号があれば単項否定 前に −，後にシンボルがあればその後 − 以外の記号までのシンボルを AND してその否定をとる	
:	上の計算結果を次のシンボルへ出力	
¥	上の計算結果の否定をシンボルへ出力	

行（思考）錯誤の結果図 5.27 のフローチャートができました．このフローは図 5.26 のフローとは意味が違い，マクロ展開時の判断，コード生成の手順をフローチャートにしたものです．マクロ展開フローはこれまでのどのマクロの説明にも使用していませんが，この **LOGIC** マクロだけは複雑な構成をとっているので直接コーディングするのは不可能でした．

というわけで，マクロ定義設計段階でフローチャートを使用したのはこのマクロが最初でした．したがって，ここに初登場となったのです．このマクロは図 5.28 の記号を演算子として解釈し，1行に記号とシンボル合計で 16 個まで書けます．カッコは使えませんが，16 個の制限内で **AND** の数，**OR** の数とも制限はありません．

図 5.27 のフローに基づいて作成したマクロ定義は図 5.29 (a)のようになりました．フローをそのままマクロに置き換えたものになっていますから，その対応の仕方を見ていただきたいと思います．フローとマクロ定義の関係さえつかめれば，複雑なマクロ設計はすべてフローチャートで設計でき，いきなりコーディングするよりミスや勘違いも大幅に減ります．

図 5.29 (b)のリストで各 **LOGIC** がどのように展開されているか確認できます．各シンボルは **PBN** 形式で **EQU** され，任意のポートの任意のビットを演算対象にできます．このリストはロジック部分だけしか書いてありませんから，実用にあたってはこの前にインタラプト前処理（レジスタ AF の **PUSH** その他，システム上必要な処理）を行い，このルーチンを抜けた後，インタラプト後処理と **RET**（または **RETI**…システム構成による）を入れなければなりません．また，このインタラプトはロジック処理に要求される応答時間より

〈図 5.29〉 図 5.27 をマクロに置き換えたロジック・シュミレータ

(a) マクロ定義

```
 1:             .Z80
 2:             .XLIST
 3:    ; ﾛｼﾞｯｸ ﾖｳ  ｻﾌﾞ ﾏｸﾛ
 4:    ;    ¦IRP ¦ﾄ¦ IFIDN ¦ﾉ ｸﾐｱﾜｾ ｶﾞ
 5:    ;    ﾀﾀﾞｼｸ ﾊﾀﾗｶﾅｲ ﾉﾃﾞ
 6:    ;    ¦IFIDN¦ ｦ ｻｹﾀ
 7:    ;
 8:    ORGEN    MACRO    OCC
 9:             IF (CC AND 15) GT 1
10:    M&OCC:
11:    OC       DEFL     OC+1
12:             ENDIF
13:             ENDM
14:    ;
15:    TOBGEN   MACRO    OC
16:             IF       MM GT 15
17:             JR       Z,M&OC
18:             ELSE
19:             JR       NZ,M&OC
20:             ENDIF
21:             ENDM
22:    ;
23:    ;ﾛｼﾞｯｸ ｺﾝﾊﾟｲﾗ ﾏｸﾛ
24:    ;
25:    LOGIC    MACRO    B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q
26:             LOCAL    TRU
27:    CC       DEFL     0
28:    MM       DEFL     16
29:             IRP X,<B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q>
30:             IRPC     T,X
31:    CK       DEFL     '&T'
32:             EXITM
33:             ENDM
34:             IF       CK EQ '¥'
35:    CC       DEFL     CC+16
36:             ELSE
37:             IF       CK EQ ':'
38:    CC       DEFL     CC+32
39:             ELSE
40:             IF       CK EQ '+'
41:             IF       MM GT 15
42:             JR       NZ,TRU
43:             ELSE
44:             JR       Z,TRU
45:             ENDIF
46:             ORGEN    %OC
47:    CC       DEFL     0
48:    MM       DEFL     16
49:             ELSE
50:             IF       CK EQ '-'
51:             IF       (MM GT 16) OR (MM EQ 1)
52:             TOBGEN   %OC
53:    MM       DEFL     0
54:             ENDIF
```

```
 55:  MM       DEFL     MM AND 15
 56:           ELSE
 57:           IF       CC GT 15
 58:           IF       MM GT 15
 59:           JR       NZ,TRU
 60:           ELSE
 61:           JR       Z,TRU
 62:           ENDIF
 63:           ORGEN    %OC
 64:           IN       A,(LOW X)
 65:           IF       CC GT 31
 66:           RES      HIGH X,A
 67:           ELSE
 68:           SET      HIGH X,A
 69:           ENDIF
 70:           JR       $+6
 71:  TRU:     IN       A,(LOW X)
 72:           IF       CC GT 31
 73:           SET      HIGH X,A
 74:           ELSE
 75:           RES      HIGH X,A
 76:           ENDIF
 77:           OUT      (LOW X),A
 78:           EXITM
 79:           ELSE
 80:           IF       MM GT 16
 81:           TOBGEN   %OC
 82:           ELSE
 83:           IF       (MM AND 15) GT 0
 84:           JR       Z,TRU
 85:           ENDIF
 86:           ENDIF
 87:  CC       DEFL     CC+1
 88:  MM       DEFL     MM+1
 89:           IN       A,(LOW X)
 90:           BIT      HIGH X,A
 91:           ENDIF
 92:           ENDIF
 93:           ENDIF
 94:           ENDIF
 95:           ENDIF
 96:           ENDM
 97:           ENDM
 98:  ;
 99:           .LIST
100:  ;ﾛｼﾞｯｸ ﾃｽﾄ
101:  ;
102:  OC       DEFL     0   ;ｵｱ ﾉ ﾄﾋﾞｻｷ ｶｳﾝﾀ
103:  ;
104:  AAA      EQU      0
105:  BBB      EQU      101H
106:  CCC      EQU      202H
107:  DDD      EQU      303H
108:  EEE      EQU      404H
109:  FFF      EQU      505H
110:  GGG      EQU      606H
```

```
111:  HHH      EQU    707H
112:  ;
113:           LOGIC  AAA BBB CCC ¥ DDD
114:           LOGIC  AAA BBB - CCC - DDD - EEE FFF : GGG
115:           LOGIC  - AAA - BBB + - CCC - DDD : EEE
116:           LOGIC  - AAA BBB + - CCC DDD ¥ EEE
117:           LOGIC  AAA BBB - CCC DDD + EEE FFF - GGG HHH : AAA
118:           END
```

(b) アセンブル・リスト

```
                          .Z80
                          .LIST
                        ;ﾛｼﾞｯｸ ﾃｽﾄ
                        ;
0000                    OC       DEFL   0  ;ｵﾌ ﾉ ﾄﾋﾞｻｷ ｶｳﾝﾀ
                        ;
0000                    AAA      EQU    0
0101                    BBB      EQU    101H
0202                    CCC      EQU    202H
0303                    DDD      EQU    303H
0404                    EEE      EQU    404H
0505                    FFF      EQU    505H
0606                    GGG      EQU    606H
0707                    HHH      EQU    707H
                        ;
                                 LOGIC  AAA BBB CCC ¥ DDD
0000'   DB 00       +            IN     A,(LOW AAA)
0002'   CB 47       +            BIT    HIGH AAA,A
0004'   28 0C       +            JR     Z,M&0
0006'   DB 01       +            IN     A,(LOW BBB)
0008'   CB 4F       +            BIT    HIGH BBB,A
000A'   28 06       +            JR     Z,M&0
000C'   DB 02       +            IN     A,(LOW CCC)
000E'   CB 57       +            BIT    HIGH CCC,A
0010'   20 06       +            JR     NZ,..0000
0012'               +       M&0:
0012'   DB 03       +            IN     A,(LOW DDD)
0014'   CB DF       +            SET    HIGH DDD,A
0016'   18 04       +            JR     $+6
0018'   DB 03       +       ..0000: IN  A,(LOW DDD)
001A'   CB 9F       +            RES    HIGH DDD,A
001C'   D3 03       +            OUT    (LOW DDD),A
                                 LOGIC  AAA BBB - CCC - DDD - EEE FFF : GGG
001E'   DB 00       +            IN     A,(LOW AAA)
0020'   CB 47       +            BIT    HIGH AAA,A
0022'   28 1E       +            JR     Z,M&1
0024'   DB 01       +            IN     A,(LOW BBB)
0026'   CB 4F       +            BIT    HIGH BBB,A
0028'   28 18       +            JR     Z,M&1
002A'   DB 02       +            IN     A,(LOW CCC)
002C'   CB 57       +            BIT    HIGH CCC,A
002E'   20 12       +            JR     NZ,M&1
0030'   DB 03       +            IN     A,(LOW DDD)
0032'   CB 5F       +            BIT    HIGH DDD,A
0034'   20 0C       +            JR     NZ,M&1
```

```
0036'   DB 04    +              IN     A,(LOW EEE)
0038'   CB 67    +              BIT    HIGH EEE,A
003A'   28 0C    +              JR     Z,..0001
003C'   DB 05    +              IN     A,(LOW FFF)
003E'   CB 6F    +              BIT    HIGH FFF,A
0040'   28 06    +              JR     Z,..0001
0042'            +      M&1:
0042'   DB 06    +              IN     A,(LOW GGG)
0044'   CB B7    +              RES    HIGH GGG,A
0046'   18 04    +              JR     $+6
0048'   DB 06    +      ..0001: IN     A,(LOW GGG)
004A'   CB F7    +              SET    HIGH GGG,A
004C'   D3 06    +              OUT    (LOW GGG),A
                                LOGIC  - AAA - BBB + - CCC - DDD : EEE
004E'   DB 00    +              IN     A,(LOW AAA)
0050'   CB 47    +              BIT    HIGH AAA,A
0052'   20 06    +              JR     NZ,M&2
0054'   DB 01    +              IN     A,(LOW BBB)
0056'   CB 4F    +              BIT    HIGH BBB,A
0058'   28 12    +              JR     Z,..0002
005A'            +      M&2:
005A'   DB 02    +              IN     A,(LOW CCC)
005C'   CB 57    +              BIT    HIGH CCC,A
005E'   20 06    +              JR     NZ,M&3
0060'   DB 03    +              IN     A,(LOW DDD)
0062'   CB 5F    +              BIT    HIGH DDD,A
0064'   28 06    +              JR     Z,..0002
0066'            +      M&3:
0066'   DB 04    +              IN     A,(LOW EEE)
0068'   CB A7    +              RES    HIGH EEE,A
006A'   18 04    +              JR     $+6
006C'   DB 04    +      ..0002: IN     A,(LOW EEE)
006E'   CB E7    +              SET    HIGH EEE,A
0070'   D3 04    +              OUT    (LOW EEE),A
                                LOGIC  - AAA BBB + - CCC DDD ¥ EEE
0072'   DB 00    +              IN     A,(LOW AAA)
0074'   CB 47    +              BIT    HIGH AAA,A
0076'   28 18    +              JR     Z,..0003
0078'   DB 01    +              IN     A,(LOW BBB)
007A'   CB 4F    +              BIT    HIGH BBB,A
007C'   28 12    +              JR     Z,..0003
007E'            +      M&4:
007E'   DB 02    +              IN     A,(LOW CCC)
0080'   CB 57    +              BIT    HIGH CCC,A
0082'   28 0C    +              JR     Z,..0003
0084'   DB 03    +              IN     A,(LOW DDD)
0086'   CB 5F    +              BIT    HIGH DDD,A
0088'   28 06    +              JR     Z,..0003
008A'            +      M&5:
008A'   DB 04    +              IN     A,(LOW EEE)
008C'   CB E7    +              SET    HIGH EEE,A
008E'   18 04    +              JR     $+6
0090'   DB 04    +      ..0003: IN     A,(LOW EEE)
0092'   CB A7    +              RES    HIGH EEE,A
0094'   D3 04    +              OUT    (LOW EEE),A
                                LOGIC  AAA BBB - CCC DDD + EEE FFF - GGG HHH : AAA
0096'   DB 00    +              IN     A,(LOW AAA)
```

```
0098'    CB 47            +            BIT    HIGH AAA,A
009A'    28 12            +            JR     Z,M&6
009C'    DB 01            +            IN     A,(LOW BBB)
009E'    CB 4F            +            BIT    HIGH BBB,A
00A0'    28 0C            +            JR     Z,M&6
00A2'    DB 02            +            IN     A,(LOW CCC)
00A4'    CB 57            +            BIT    HIGH CCC,A
00A6'    28 24            +            JR     Z,..0004
00A8'    DB 03            +            IN     A,(LOW DDD)
00AA'    CB 5F            +            BIT    HIGH DDD,A
00AC'    28 1E            +            JR     Z,..0004
00AE'                     +    M&6:
00AE'    DB 04            +            IN     A,(LOW EEE)
00B0'    CB 67            +            BIT    HIGH EEE,A
00B2'    28 12            +            JR     Z,M&7
00B4'    DB 05            +            IN     A,(LOW FFF)
00B6'    CB 6F            +            BIT    HIGH FFF,A
00B8'    28 0C            +            JR     Z,M&7
00BA'    DB 06            +            IN     A,(LOW GGG)
00BC'    CB 77            +            BIT    HIGH GGG,A
00BE'    28 0C            +            JR     Z,..0004
00C0'    DB 07            +            IN     A,(LOW HHH)
00C2'    CB 7F            +            BIT    HIGH HHH,A
00C4'    28 06            +            JR     Z,..0004
00C6'                     +    M&7:
00C6'    DB 00            +            IN     A,(LOW AAA)
00C8'    CB 87            +            RES    HIGH AAA,A
00CA'    18 04            +            JR     $+6
00CC'    DB 00            +    ..0004: IN     A,(LOW AAA)
00CE'    CB C7            +            SET    HIGH AAA,A
00D0'    D3 00            +            OUT    (LOW AAA),A
                                       END
```

短い間隔で起動されなければなりません.

ロジック処理が多くても，また5.2節のソフトウェア・タイマと組み合わせてもハードウェア上のインタラプト・タイマは1チャネルしかいりません．このロジックの処理時間はシンボル1個当たり **IN** と **BIT** で19クロック，それに出力するための **OUT** が11クロックですから，

行数×11＋シンボル数×19 →クロック数

となります．ちなみに図5.29の例では30シンボルと5行ですから625クロック．4 MHzのCPUなら156.25 μsで完了します(他に **JR** が入りますが，条件によって見ないビットが出てきますから，平均的にはこの値より速くなります)．この時間は，もし **CPU** 実行時間の半分をインタラプトに費やしてもよいとして，10 msのインタラプト(ロジックの応答時間を10msとする）を使用すると，この例のような処理が31回もできることになりま

〈図5.30〉図5.29の式の変形

113行	$\overline{(A \cdot B \cdot C)} \rightarrow D$	
114行	$A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot \overline{(E \cdot F)} \rightarrow G$	
115行	$(\overline{A} \cdot \overline{B}) + (\overline{C} \cdot \overline{D}) \rightarrow E$	$\overline{(A + B) \cdot (C + D)}$
116行	$\overline{(\overline{A} \cdot B) + (\overline{C} \cdot D)} \rightarrow E$	$(A + \overline{B}) \cdot (C + \overline{D})$
117行	$A \cdot B \cdot \overline{(C \cdot D)} + E \cdot F \cdot \overline{(G \cdot H)} \rightarrow A$	

（**A**は **AAA** の意味．**B,C**…… も同じ）

す．また5.2節のソフト・タイマと共用してさらに半分ずつに分けたとしてもこの例の程度の処理が75行も使え，ソフト・タイマは50チャネル使用可能です．

この方式では組み込みロジックがどんなに変わっても，ハードウェアはポートの数さえ足りれば特殊な変更を全く要しません．しかも，メイン・ルーチンの流れと無関係に片手間に処理できます．**LOGIC** で書いた論理式を回路で作って組み込んだつもりで使用できます．

図5.30は図5.29の各式を数学流の論理式で示し，その変形を試みたものです．この書式でもほとんどの要求に応えられることがわかると思います．このマクロは当然ロジック処理のみを扱う用途にも使用可能です．また，インタラプトの飛び先を変えて全く異なるロジック処理に瞬時に切り換えるなど，コンピュータ・ロジックならではの応用も可能です．

5.7 BPU用1文字ロジック・マクロ ——シンボル間のブランクも不要に

前節のマクロ **LOGIC** の考え方で，BPU用のクロス・マクロを作るにあたって，BPUではポートの数が32までなのでこれを何とか1文字で表せないかと考えました．もし1文字で表せるとすれば，**IRPC** で各文字を分離してその文字を解釈する方法で，パラメータの区切り記号が全くなくても展開可能なはずです．

ところで，アルファベットは26文字ですから32のポートを表しきれません．そこで，大文字と小文字を区別して使用する方法を採りました．さらに，数字も1文字のみでシンボルを表せるようにしました(もちろんアセンブラはこのようなシンボルを解釈できません．すべてマクロ展開時にしかるべき処理をします)．

また，もともとロジック専用であるBPU用としては前節のロジックの機能だけでは不

十分なのでいくつかの機能を追加しました.でき上がったマクロの仕様は図5.32のようになりました.

5.7.1 文字間を詰めて書け，ブランクは無視される

IRPC を使用して文字ごとに解釈するので途中のブランクは無視されます．マクロが実パラメータを受け取るのに仮パラメータを6個しか用意していないので，文字列の数として6個まで，すなわち途中のブランクは5回まで許されます．6回目が現れた時はその行は終了と見なされます．この数はマクロ定義で変更できます．ブランクは書式上，見やすいように入れて使います.

5.7.2 ひとつの式が行を越えてもよい

1文字ずつの展開でブランクを無視したついでに，行を越えても式が解釈できるようにしました．1行の文字数はアセンブラによって(M80では132文字までに)制限されますが，行を越えることによってどんなに長い式でも書けます

5.7.3 空の式を解釈

式の結果を出力したりするときに，いきなり出力する部分やジャンプの部分が書かれることを許します．空の式は成立している(論理値＝1)ものと見なされます．各ポートの初期設定も **OUT** 命令を書かないでできるようになります.

:A (Aに1を出力)

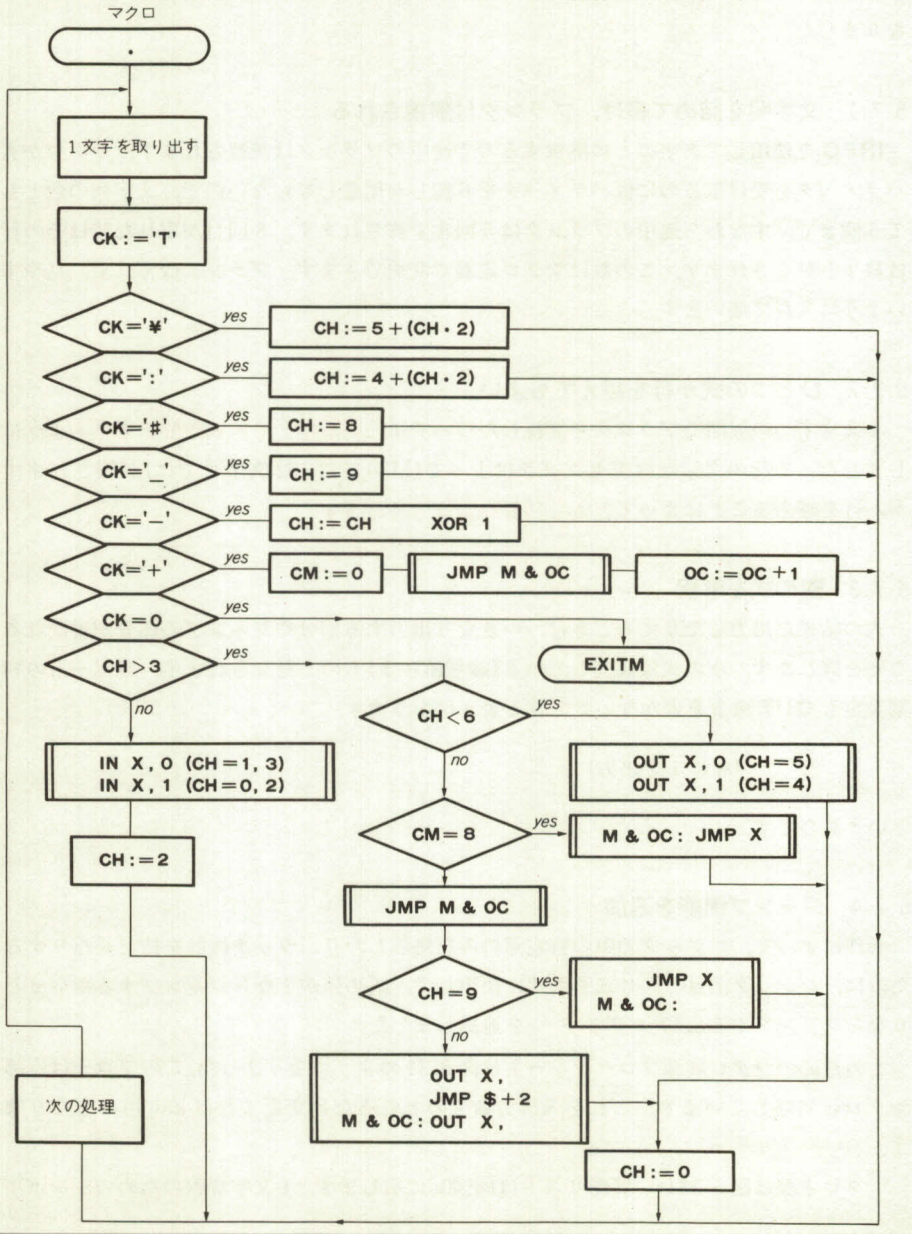
という具合です.

5.7.4 ジャンプ機能を追加

条件によって，ロジックの中の特定部のみを処理したり，ラッチ機能を持たせたりするために，ロジック演算と同じ式を条件に使用して，式の値が1ならジャンプする命令＃と0ならジャンプする_(アンダ・バー)を追加しました.

このためのマクロ展開フローチャートは図5.31のようになりました.このフローは内部マクロは省略しています．これを入出力命令のところだけ変更すればZ80用にもなります.

マクロ定義は図5.33(a)，展開リストは同図(b)に示します．1文字解釈のためのシンボル

〈図 5.31〉 BPU 用1文字ロジック・マクロの展開フローチャート

〈図5.32〉 BPU用1文字ロジック・マクロの仕様

シンボル…………	各1文字がひとつのシンボルを表す． 大文字はそれ自身を，小文字はその文字を2個ならべたものを， 数字はJのあとにその数字を付加したものをそれぞれ表す．
記号なし…………	シンボル間に記号がなければANDやORより優先．
ブランク…………	すべて無視される（ただし1行の中で5個まで）．
＋ …………………	OR
－ …………………	前に－，後に記号があれば単項否定 前に－，後にシンボルが続けば，その後－以外の記号までのシンボルをANDしてその否定をとる．
： …………………	上の計算結果を次のシンボルへ出力．
¥ …………………	上の計算結果の否定を次のシンボルへ出力．
# …………………	上の計算結果が1なら次のシンボルへジャンプ．
_ …………………	上の計算結果が0なら次のシンボルへジャンプ． （：，¥，#，_の各文字は，その次の一文字でひとつの処理を終わる．処理は何行かにわたってもよい．）
空の式……………	式がなく，：，¥，#，_が現れた時は式の値は1とする．

合成マクロに **SYMGEN** が内部マクロとして追加されています．論理式の評価法は前節のものと全く同じで十分実用になります．BPUでは基本メモリ128バイトまでなので，すべて **ASEG** でアセンブルしています．

〈図5.33〉 1文字で変数を表すロジック用マクロ

(a) ソース・リスト

```
 1:  ;   |BPU|ﾉﾀﾒﾉ ﾛｼﾞｯｸ ｼｮﾘ
 2:  ;   ﾎﾟｰﾄ ﾄ ﾗﾍﾞﾙ ｦ ｽﾍﾞﾃ |1|ﾓｼﾞ ﾄ ｼﾃ
 3:  ;   ﾖﾘ ﾛﾝﾘｼｷ ﾗｼｸ ﾐｾﾙ
 4:  ;   oooooooooooooooooooooooooooooooooooooooo
 5:  ;
 6:  ; ﾛｼﾞｯｸ ﾖｳ   ｻﾌﾞ ﾏｸﾛ
 7:  ;
 8:  LABGEN   MACRO    OCC
 9:  M&OCC:
10:  OC       DEFL     OC+1
11:           ENDM
12:  ;
13:  TOBGEN   MACRO    QQ
14:           DB       M&QQ OR 128
15:           ENDM
16:  ;
```

```
17:  ;ｵｰﾓｼﾞ ｶ ｺﾓｼﾞ ｶ ｽｳｼﾞ ｦ ﾊﾝﾃｲ
18:  ;!A --> A ; a --> AA ; 0 --> J0
19:  ;
20:  SYMGEN  MACRO   X,Y
21:          IF      '&X' GT 47 AND '&X' LT 58
22:          DB      J&X OR Y
23:          ELSE
24:          IF      '&X' GT 96 AND '&X' LT 123
25:          DB      X&X OR Y
26:          ELSE
27:          DB      X OR Y
28:          ENDIF
29:          ENDIF
30:          ENDM
31:  ;
32:  ;
33:  ;ﾛｼﾞｯｸ ｺﾝﾊﾟｲﾗ ﾏｸﾛ
34:  ;
35:          MACRO   B,C,D,E,F,G
36:          IRP W,<B,C,D,E,F,G>
37:          IRPC    X,W
38:  CK      DEFL    '&X'
39:          IF      CK EQ '¥'
40:  CH      DEFL    5+(CH AND 2)
41:          ELSE
42:          IF      CK EQ ':'
43:  CH      DEFL    4+(CH AND 2)
44:          ELSE
45:          IF      CK EQ '#'
46:  CH      DEFL    8
47:          ELSE
48:          IF      CK EQ '_'
49:  CH      DEFL    9
50:          ELSE
51:          IF      CK EQ '-'
52:  CH      DEFL    CH XOR 1
53:          ELSE
54:          IF      CK EQ '+'
55:  CH      DEFL    0
56:          TOBGEN  %OC
57:          ELSE
58:          IF      CK EQ 0
59:          EXITM
60:          ELSE
61:          IF      CH GT 3
62:          IF      CH LT 6
63:          SYMGEN  X,%(32*(CH+1 AND 1))
64:          ELSE
65:          IF      CH EQ 8
66:          LABGEN  %OC
67:          SYMGEN  X,128
68:          ELSE
69:          TOBGEN  %OC
70:          IF      CH EQ 9
71:          SYMGEN  X,128
72:          LABGEN  %OC
73:          ELSE
```

```
 74:           SYMGEN  X,%(32*(CH AND 1))
 75:           DB      $+2 OR 128
 76:           LABGEN  %OC
 77:           SYMGEN  X,%(32*(CH+1 AND 1))
 78:           ENDIF
 79:           ENDIF
 80:           ENDIF
 81:  CH       DEFL    0
 82:           ELSE
 83:           SYMGEN  X,%(64 OR 32*(CH+1 AND 1))
 84:  CH       DEFL    2
 85:           ENDIF
 86:           ENDIF
 87:           ENDIF
 88:           ENDIF
 89:           ENDIF
 90:           ENDIF
 91:           ENDIF
 92:           ENDIF
 93:           ENDM
 94:           ENDM
 95:           ENDM
 96:  ;
 97:  ;ロジ゛ック テスト
 98:  ;
 99:  OC       DEFL    0   ;トビ゛サキ カウンタ
100:  CH       DEFL    0   ;シキ ノ ジ゛ョウタイ !FLAG
101:  ;
102:           ASEG        ;アブ゛ソルート ニ コテイ スル
103:  ;
104:  A        EQU     1
105:  B        EQU     2
106:  C        EQU     3
107:  D        EQU     4
108:  E        EQU     5
109:  F        EQU     6
110:  G        EQU     7
111:  H        EQU     8
112:  aa       EQU     17
113:  bb       EQU     18
114:  cc       EQU     19
115:  dd       EQU     20
116:  ee       EQU     21
117:  ff       EQU     22
118:  gg       EQU     23
119:  hh       EQU     24
120:  ;
121:  TT:      . :a:b:c¥D¥E¥F
122:  Q:       . AB+CD:E A-B+C-D¥E
123:           . -f-g#Q ¥H
124:  J1:      . AB#1
125:           . -A-B-C_Q
126:           . A+B+C#t #Q
127:           END
```

(b) アセンブル・リスト

```
;   BPUﾉﾀﾒﾉ ﾛｼﾞｯｸ ｼｮﾘ
;   ﾎﾟｰﾄ ﾄ ﾗﾍﾞﾙ ｦ ｽﾍﾞﾃ 1ﾓｼﾞ ﾄ ｼﾃ
;   ﾖﾘ ﾛﾝﾘｼｷ ﾗｼｸ ﾐｾﾙ
;   ••••••••••••••••••••••••••••••••••••••••
;
; ﾛｼﾞｯｸ ﾖｳ  ｻﾌﾞ ﾏｸﾛ
;
LABGEN  MACRO   OCC
M&OCC:
OC      DEFL    OC+1
        ENDM
;
TOBGEN  MACRO   QQ
        DB      M&QQ OR 128
        ENDM
;
;ｵｰﾓｼﾞ ｶ ｺﾓｼﾞ ｶ ｽｳｼﾞ ｦ ﾊﾝﾃｲ
;A --> A ; a --> AA ; 0 --> J0
;
SYMGEN  MACRO   X,Y
        IF      '&X' GT 47 AND '&X' LT 58
        DB      J&X OR Y
        ELSE
        IF      '&X' GT 96 AND '&X' LT 123
        DB      X&X OR Y
        ELSE
        DB      X OR Y
        ENDIF
        ENDIF
        ENDM
;
;
;ﾛｼﾞｯｸ ｺﾝﾊﾟｲﾗ ﾏｸﾛ
;
        MACRO   B,C,D,E,F,G
        IRP W,<B,C,D,E,F,G>
        IRPC    X,W
CK      DEFL    '&X'
        IF      CK EQ '¥'
CH      DEFL    5+(CH AND 2)
        ELSE
        IF      CK EQ ':'
CH      DEFL    4+(CH AND 2)
        ELSE
        IF      CK EQ '#'
CH      DEFL    8
        ELSE
        IF      CK EQ '_'
CH      DEFL    9
        ELSE
        IF      CK EQ '-'
CH      DEFL    CH XOR 1
        ELSE
        IF      CK EQ '+'
```

```
                CH      DEFL    0
                        TOBGEN  %OC
                                ELSE
                                IF      CK EQ 0
                                EXITM
                                ELSE
                                IF      CH GT 3
                                IF      CH LT 6
                                SYMGEN  X,%(32*(CH+1 AND 1))
                                ELSE
                                IF      CH EQ 8
                                LABGEN  %OC
                                SYMGEN  X,128
                                ELSE
                                TOBGEN  %OC
                                IF      CH EQ 9
                                SYMGEN  X,128
                                LABGEN  %OC
                                ELSE
                                SYMGEN  X,%(32*(CH AND 1))
                                DB      $+2 OR 128
                                LABGEN  %OC
                                SYMGEN  X,%(32*(CH+1 AND 1))
                                ENDIF
                                ENDIF
                                ENDIF
                        CH      DEFL    0
                                ELSE
                                SYMGEN  X,%(64 OR 32*(CH+1 AND 1))
                        CH      DEFL    2
                                ENDIF
                                ENDIF
                                ENDIF
                                ENDIF
                                ENDIF
                                ENDIF
                                ENDIF
                                ENDIF
                                ENDM
                                ENDM
                                ENDM
                        ;
                        ;ﾛｼﾞｯｸ ﾃｽﾄ
                        ;
0000                    OC      DEFL    0   ;ﾄﾋﾞｻｷ ｶｳﾝﾀ
0000                    CH      DEFL    0   ;ｼｷ ﾉ ｼﾞｮｳﾀｲ FLAG
                        ;
0000                            ASEG        ;ｱﾌﾞｿﾙｰﾄ ﾆ ｺﾃｲ ｽﾙ
                        ;
0001                    A       EQU     1
0002                    B       EQU     2
0003                    C       EQU     3
0004                    D       EQU     4
0005                    E       EQU     5
0006                    F       EQU     6
0007                    G       EQU     7
```

```
0008                    H       EQU    8
0011                    aa      EQU    17
0012                    bb      EQU    18
0013                    cc      EQU    19
0014                    dd      EQU    20
0015                    ee      EQU    21
0016                    ff      EQU    22
0017                    gg      EQU    23
0018                    hh      EQU    24
                        ;
0000                    TT:     . :a:b:c¥D¥E¥F
0000    31          +           DB     a&a OR 32
0001    32          +           DB     b&b OR 32
0002    33          +           DB     c&c OR 32
0003    04          +           DB     D OR 0
0004    05          +           DB     E OR 0
0005    06          +           DB     F OR 0
0006                    Q:      . AB+CD:E A-B+C-D¥E
0006    61          +           DB     A OR 96
0007    62          +           DB     B OR 96
0008    8E          +           DB     M&0 OR 128
0009    63          +           DB     C OR 96
000A    64          +           DB     D OR 96
000B    8E          +           DB     M&0 OR 128
000C    05          +           DB     E OR 0
000D    8F          +           DB     $+2 OR 128
000E                +   M&0:
000E    25          +           DB     E OR 32
000F    61          +           DB     A OR 96
0010    42          +           DB     B OR 64
0011    97          +           DB     M&1 OR 128
0012    63          +           DB     C OR 96
0013    44          +           DB     D OR 64
0014    97          +           DB     M&1 OR 128
0015    25          +           DB     E OR 32
0016    98          +           DB     $+2 OR 128
0017                +   M&1:
0017    05          +           DB     E OR 0
                                . -f-g#Q ¥H
0018    56          +           DB     f&f OR 64
0019    57          +           DB     g&g OR 64
001A                +   M&2:
001A    86          +           DB     Q OR 128
001B    08          +           DB     H OR 0
001C                    J1:     . AB#1
001C    61          +           DB     A OR 96
001D    62          +           DB     B OR 96
001E                +   M&3:
001E    9C          +           DB     J&1 OR 128
                                . -A-B-C_Q
001F    41          +           DB     A OR 64
0020    42          +           DB     B OR 64
0021    43          +           DB     C OR 64
0022    A4          +           DB     M&4 OR 128
0023    86          +           DB     Q OR 128
0024                +   M&4:
```

```
                                   . A+B+C#t #Q
0024    61          +              DB      A OR 96
0025    A9          +              DB      M&5 OR 128
0026    62          +              DB      B OR 96
0027    A9          +              DB      M&5 OR 128
0028    63          +              DB      C OR 96
0029                +        M&5:
0029    80          +              DB      t&t OR 128
002A                +        M&6:
002A    86          +              DB      Q OR 128
                                   END
```

第 6 章

アセンブラのソフトウェア開発における高級言語の利用

1.5 節でも述べましたが，ソフトウェア開発はできる限り高級言語を利用した方が得策です．少なくとも今後は高級言語もさらに多様化してくると予想され，現在のところは応用しきれないという分野にも適用可能な言語が登場することは十分考えられます．

しかし，将来のことよりも，今どうするかが問題だという考え方も成り立ちます．それでは現時点でどうしてもアセンブラによる記述によらざるを得ない分野，システムについて高級言語の利用価値はないものでしょうか．

これには大きく分けて 3 種類の利用法が考えられます．第一にアセンブラに入力するソース・ファイルを作る段階以前に使用する方法．第二にアセンブラとリンクしてプログラムの一部分を高級言語で書く方法，そして第三にはアセンブラの出力リストを見やすくするなど，アセンブル以後の段階で利用する方法です．これら 3 つの方法は各々独立で相互関係を持っていませんから，各段階での利用を混合することも自由です．

本章ではこの三種の利用法の実例を紹介します．使用する高級言語は Pascal/MT＋という名称のコンパイラで，ディジタル・リサーチ社の製品です．

6.1 リスト上のカナ文字復元プログラム

4.3 節ですでに，カナ文字をコメント，タイトル，そしてリテラル・データに使用する方法を書きました．また，本書で使用しているリストのほとんどがカナ文字を使用しています．これらはすべてここに紹介するカナ文字復元プログラムによって M80 の出力ファイル.**PRN** を変換したものです．

このプログラムは **KANA** と名付けました(図 6.1)．動作の基本は，起動後コンソールからファイル名を入力し，そのファイル名の後に.**PRN** を付加してファイルを読み取り，変換されたファイルを同じファイル名で.**PRM**と変えて出力します．.**PRN** はそのまま残ります．また，このままでは 1 回の起動でいくつかのファイルを変換することはできません．

〈図6.1〉 カナ文字を復元するプログラム

```
 1:
 2:   PROGRAM KANA ;
 3:   VAR FI,FO : FILE OF CHAR ;
 4:       CH : CHAR ;
 5:       KANA : BOOLEAN ;
 6:       AH,BH : BYTE ;
 7:       CF : STRING[13] ;
 8:       RSLT : INTEGER  ;
 9:
10:   PROCEDURE MODFY (EC:BYTE ; KI:BOOLEAN) ;(*ｶﾅ ｺｰﾄﾞ ﾆ ﾍﾝｶﾝ*)
11:   BEGIN
12:     KANA := KI ;
13:     REPEAT
14:       READ (FI,CH) ;
15:       BH := CH ;
16:       IF CH = '¦' THEN
17:         KANA := NOT KANA
18:       ELSE
19:         BEGIN ;
20:           IF (NOT KANA) OR (BH=13) OR (BH=EC) THEN
21:             WRITE (FO,CH)
22:           ELSE
23:             BEGIN
24:               BH := BH!128 ;
25:               CH := BH ;
26:               WRITE (FO,CH)
27:             END
28:         END
29:     UNTIL (BH=13) OR (BH=EC) OR EOF
30:   END ;
31:   BEGIN
32:     WRITELN ;
33:     WRITE(' ﾌｧｲﾙﾒｰ ? ') ;
34:     READ(CF) ;
35:     IF CF<>'Q' THEN
36:       BEGIN
37:         OPEN(FI,CONCAT(CF,'.PRN'),RSLT);
38:         IF RSLT=255 THEN
39:           WRITELN('ｿﾉ ﾌﾜｲﾙ ﾜ ﾖﾒﾏﾍﾝﾜｧ ')
40:         ELSE
41:           BEGIN
42:             ASSIGN(FO,CONCAT(CF,'.PRM')) ;
43:             REWRITE (FO) ;
44:             CH := ' ' ;
45:             REPEAT
46:               AH := CH ;
47:               READ(FI,CH) ;
48:               WRITE(FO,CH) ;
49:               IF (AH<47) OR ((AH>57) AND (AH<64)) THEN
50:                 CASE CH OF
51:                   ';' : MODFY (13,TRUE) ;
52:                   '"' : MODFY (34,TRUE) ;
53:                   '''': MODFY (39,FALSE)
54:                 END
```

```
55:                 UNTIL EOF(FI) ;
56:                 CLOSE(FI,RSLT) ;
57:                 CLOSE(FO,RSLT) ;
58:                 IF RSLT = 255 THEN
59:                   WRITELN('CLOSE ERROR ')
60:                 ELSE
61:                   WRITELN('                    オワリマシタ  ')
62:               END
63:         END
64:   END.
65:
```

毎回CP/Mに戻ります．

カナ文字の復元は次のような手順によっています．

①セミコロン(；)を検出したら，その行の終わり(リターン・コード)までをカナ変換対象とする．

②ダブル・コート(")を検出したら，再びダブル・コートが現れるまでをカナ変換対象とする．

③シングル・コート(′)を検出したら，再びシングル・コートが現れるまでをカナ変換対象とする．

④ただし，検出文字の直前の文字が数字やアルファベットの場合は変換作業に入らない．

⑤カナ変換にあたっては，セミコロンとダブル・コートはカナ文字から始まるものとし，シングル・コートではアルファ数字から始まるものとする．

⑥カナ変換中にコード**7CH**(¦)を検出したら，カナとアルファ数字の状態を逆にする(カナ・シフト)．

⑦カナ変換作業は行の終わりを越えない．

これによって変換されたアセンブル・リストは，**MESAG**(4.3節，5.1節)マクロによってコンソールに出力されるメッセージと一致します．図6.1ではファイル**.PRM**を作るようになっていますが，直接プリンタへ出力するようにもできます．それには42行と43行を，

OPEN (FO, 'LST:', RSLT)

の1行に変更すればよいでしょう．

6.2 クロス・アセンブラ用リフォーマッタ

5.5節でBPUのクロス・アセンブラをM80のマクロ定義で実現することについて解説しました．そして，M80のアセンブル・リストそのままではBPUのアセンブラとして見るには適していないので図5.22のように変換しました．この変換によって，マクロ呼び出しの行に展開時のロケーションおよびそのときのPCモードと，生成された機械語の16進表示が得られます．

この変換機能はクロス・アセンブラはもちろん，通常のマクロ呼び出しを多用するプログラムにとっても利用価値があります．というのは，マクロ・アセンブラの利用価値のひとつに，大きなプログラムでも小さなリストで済む点があり，これは展開リストを出力しない場合に限られます．さりとて，マクロ呼び出し行だけのリストを**.SALL**指定で取り出した場合，ソース･プログラムを見ているのと何ら変わりません．図6.2(a)は図5.16(b)のリストを**.SALL**で取り直したマクロ呼び出し部分です．マクロ行を示す＋の記号がついている以外はソース・ファイルと同じです．これではデバッグの時にアドレスも機械語コードも全くわからず，ブレークをかけたりPCを指定してスタートさせたりすることができません．

せっかくのマクロ命令ですから，マクロのレベルで(機械語のステップでなく)書き，読み，考えたい．しかし，ロケーションもわからないのではデバッグに困る．そこで，このリスト変換用プログラムが活きてくるのです．これでリフォーマットしたのが図6.2(b)です．行数は変わらずに，ロケーションと最初の展開された機械語コードがわかります．

ロケーションだけでもないよりはよいのですが，その内容もわかると安心感が違います．現実に間違いも減小します．リロケータブルのアセンブラではロケーションがデバッグ時の絶対番地と一致しませんから，通常はグローバル・シンボルの番地との差を計算してアドレスを求めます．このとき，しかるべき番地の内容がわかると，計算に間違いがあってもすぐにわかります．ロケーション情報だけだとよほど慎重に計算しないと暴走させたりします．

このプログラムには**FHEN**(ファイル変換のつもり)という名前を付けました(図6.3)．動作としては，ファイル名を指定するまでは**KANA**と同じです．また，変換されるファイル名なども同じになっています．変換作業は大きく分けて行内のフォーマット変換とページ関係のフォーマット変換の2つになります．

行内フォーマット変換作業は次の手順で行います．

〈図 6.2(a)〉 **.SALL**ではリストもソースも同じ

```
                          .SALL
          +               .1        GG AA + CC
          +               .1        FF BB - CC
          +               .1        GG DD * EE
          +               .1        HH EE / DD
          +               .1        FF AA & BB
          +               .1        GG EE ! CC
          +               .1        HH DD ~ BB
          +               .1        GG AA ¥ DD
                          END
```

〈図 6.2(b)〉 **FHEN**を通すとリストらしくなる

```
0018'   3A 0010'    ¥ ←FHENを通った目印 .1      GG AA + CC
0022'   3A 0011'    ¥                 .1      FF BB - CC
002C'   3A 0013'    ¥                 .1      GG DD * EE
0038'   3A 0014'    ¥                 .1      HH EE / DD
0044'   3A 0010'    ¥                 .1      FF AA & BB
004E'   3A 0014'    ¥                 .1      GG EE ! CC
0058'   3A 0013'    ¥                 .1      HH DD ~ BB
0062'   3A 0010'    ¥                 .1      GG AA ¥ DD
                                      END
```

〈図 6.3〉 ファイル変換プログラム **FHEN**

```
 1:
 2:  PROGRAM FHEN;
 3:  VAR FI,FO : TEXT ;
 4:      S0,S1,S2 : STRING[127] ;
 5:      CF : STRING[13] ;
 6:      RSLT,PN,LC,L : INTEGER  ;
 7:
 8:  PROCEDURE TITLE (*ﾀｲﾄﾙ ｶｷﾀﾞｼ *) ;
 9:  BEGIN
10:    WRITELN(FO,S0,PN);
11:    PN := PN+1 ;
12:    LC := 0
13:  END ;
14:
15:  PROCEDURE RDFI (* ﾌｧｲﾙ ﾖﾐﾄﾘ･ ﾍｯﾀﾞ ﾖﾐﾄﾊﾞｼ *) ;
16:  BEGIN
17:    READLN(FI,S2) ;
18:    IF S2[1]=CHR(12) THEN
19:      BEGIN
20:        L := LENGTH(S2) ;
21:        IF S2[L]='S' THEN
22:          BEGIN
23:            WRITELN(FO,S1) ;
```

```
24:              WRITELN(FO,S2) ;
25:              REPEAT
26:                READLN(FI,S1) ;
27:                WRITELN(FO,S1)
28:              UNTIL EOF(FI)
29:            END
30:          ELSE
31:            BEGIN
32:              READLN(FI,S2);
33:              READLN(FI,S2);
34:              READLN(FI,S2)
35:            END
36:        END
37:    END ;
38:    BEGIN
39:      WRITELN ;
40:      WRITE('   ﾌｧｲﾙﾒｲ   ?') ;
41:      READ (CF) ;
42:      IF CF<>'Q' THEN
43:        BEGIN
44:          OPEN(FI,CONCAT(CF,'.PRN'),RSLT);
45:          IF RSLT=255 THEN
46:            WRITELN('  ｺﾞｼﾃｲ ﾉ ﾌｧｲﾙ ﾊ ﾖﾐﾀﾞｼ ﾃﾞｷﾏｾﾝ')
47:          ELSE
48:            BEGIN
49:              ASSIGN(FO,CONCAT(CF,'.PRM')) ;
50:              REWRITE (FO) ;
51:              READLN(FI,SØ) ;
52:              L := LENGTH(SØ) ;
53:              WRITE(L) ;
54:              SØ := CONCAT(COPY(SØ,2,L-1Ø),'ﾍﾟｰｼﾞ',CHR(9)) ;
55:              WRITELN(SØ) ;
56:              PN := 1 ;
57:              TITLE ;
58:              READLN(FI,S1) ;
59:              READLN(FI,S1) ;
60:              READLN(FI,S1) ;
61:              REPEAT
62:                RDFI ;
63:                IF S2[27]='+' THEN
64:                  BEGIN
65:                    L := LENGTH(S1)-27 ;
66:                    IF L<1 THEN
67:                      L := Ø ;
68:            S1 := CONCAT(COPY(S2,1,26),'¥',COPY(S1,28,L)) ;
69:                    REPEAT
70:                      RDFI
71:                    UNTIL S2[27]<>'+'
72:                  END ;
```

```
73:                 WRITELN(FO,S1) ;
74:                 LC := LC+1 ;
75:                 IF LC > 49 THEN
76:                   BEGIN
77:                     WRITE(FO,CHR(12)) ;
78:                     TITLE
79:                   END ;
80:                 S1 := S2
81:               UNTIL EOF(FI) ;
82:               CLOSE(FI,RSLT) ;
83:               CLOSE(FO,RSLT) ;
84:               WRITELN ;
85:               WRITELN('       ｵﾜﾘﾏｼﾀ')
86:             END
87:         END
88:     END.
```

①第27カラム目に＋の記号(マクロ展開行を示す記号)が書かれている行を捜す.

②その行の直前の行より前は，すべてそのまま出力ファイルに書き出す.

③その行の直前の行に,＋の記号を含む行の第26カラム目までをすげ替える(すげ替える前はすべて空白のはず).

④その行以後は第27カラム目に＋の記号を含む行が続く限り読み捨てる.

この作業は，マクロ呼び出し行を捜してマクロ展開の最初の行と合成した1行に作り変えることに相当します．原理としてはこれだけの作業で一応の動作はしてくれますが，実際には2つの問題点が残ります.

共に改ページ（**FF**）動作に関係するものですが，その第一はマクロ展開された行はすべてなくなってしまうので改ページのコードと，実際に1ページあたりに書かれる行数とが合わなくなってしまうことです．これではリスト印字時間と文字数は削減できますが，ページ数は減りませんし，見やすくもなりません.

第二の問題は，手順③のその直前の行の意味です．この手順としては，暗にアセンブルされた結果に作られた行を指していますが，現実には各ページにタイトル行とページ番号が改ページごとに出力されています．また，その後に2行の空白行が送られています．これは＋を含まない直前の行と判断されます．タイトル行と空白の2行は判断の対象としてはまずいのです.

これら2つの問題点を解決するには，改ページと空白行を取り除いて判断，変換を行い，結果のリストに再び改ページ記号とタイトル文字を入れるようにしなければなりません.

新たな改ページ記号は変換結果の50行に1回の割で付加します．また，タイトル行は，最初の行の情報を保存しておいて毎回同じものを出力し，ページ数のみカウントします．

リストの最後のシンボル表のページはタイトル行がSという文字で終わるのを検知して，それ以後はすべての変換作業を止めて入力ファイルから出力ファイルへ筒抜けにします．この **FHEN** を使えば，マクロの利用価値が倍加します．

このプログラムを **KANA** 同様，直接プリントするように変更可能です．変更箇所は49行，50行です．

ファイルの変換プログラムはアセンブラを使用しても書けますが高級言語を使った方が圧倒的に有利です．ここに取り上げた **KANA, FHEN** は大きな処理ではないので，BASICでもFORTRANでも同様の作業ができると思います．もちろん，インタプリタでも構いません．ただし，CP/Mのファイルを扱えるものでなければ困ります．

6.3 高速BCD変換プログラム——変換表をPascalで作成

ディジタル化した測定器が普及して，多くの測定器が目で読むだけでなく外部へ測定値を取り出せるようになってきました．これらのほとんどが表示値の関係上，BCDコードまたは10進数の文字コードで測定値を出しています．一方，コンピュータの方は少し複雑な処理になるとBCDでは遅すぎたり，データ・ビット数が多くなりすぎて不利です．浮動小数点演算用のICもバイナリ・コードでないと処理できません．

そこで，BCDをバイナリに変換したり，バイナリをBCDに変換する作業を高速に処理したいという要求が出てきます．通常のシステムではBCDや10進は人間とのインターフェースが目的なので高速性はあまり必要ありませんが，測定器を何台か使って逐次入ってくるデータを処理するとなると話が違ってきます．

BCDの5桁をバイナリの16ビットに変換するのにかかる時間は，数千クロックが普通です．バイナリからBCDの変換はもっと遅いでしょう．そこでもっと速い変換が要求された場合に使える高速変換プログラムを作りました．

その原理は図6.4のように各4ビットごとの桁の重みにその桁の数値をかけたものをあらかじめ，桁数×($N-1$)…(Nは進法の基数)の表として用意しておき，入力された数の各桁の値でインデクシングして目的の値を取り出し，それを合計するという方法です．たとえば，10進の3741は1の桁の1と10の桁の4，100の桁の7…と加算して行き，全桁終わった時の合計が変換結果になります．加算はすべて最終結果のビット数で行い，表

〈図6.4〉高速 BCD↔BIN 変換の原理

桁の値	1の桁	10の桁	100の桁	1000の桁
1	0001	000A	0064	03E8
2	0002	0014	00C8	07D0
3	0003	001E	012C	0BB8
4	0004	0028	0190	0FA0
5	0005	0032	01F4	1388
6	0006	003C	0258	1770
7	0007	0046	02BC	1B58
8	0008	0050	0320	1F40
9	0009	005A	0384	2328

(a) BCD→16進

```
(3741)10 ──→  0BB8H
              02BCH
              0028H
          +)      1
          ─────────
              0E9DH

20BEH ──→     8192
                 0
              0176
          +)  0014
          ─────────
            (8382)10
```

桁の値	1の桁	16の桁	256の桁	4096の桁
1	0001	0016	0256	4096
2	0002	0032	0512	8192
3	0003	0048	0768	—
4	0004	0064	1024	—
5	0005	0080	1280	—
6	0006	0096	1536	—
7	0007	0112	1792	—
8	0008	0128	2048	—
9	0009	0144	2304	—
A	0010	0160	2560	—
B	0011	0176	2816	—
C	0012	0192	3072	—
D	0013	0208	3328	—
E	0014	0224	3584	—
F	0015	0240	3840	—

(b) 16進→BCD

もすべて加算と同じビット数で作っておきます.

BCDからバイナリの場合は10進で表を引いて，バイナリで加算し，バイナリからBCDのときは，16進で表を引いて10進で加算します．どちらの変換も全く同じ考え方で割り算や引き算が使用されない点に注意してください.

図6.4では桁数が足りないので，8桁のBCDを扱えるように大きな表(データ数72個，各データ32ビット)を作り，32ビットで演算するように構成したマクロ定義と呼び出し例が図6.5です．このプログラムは8桁のBCDをバイナリに変換するのに1546クロック，バイナリから8桁のBCDに変換するのに1602クロックを要します.

プログラムの大きさは表を含めて約1.3Kバイト，700バイトが表で埋まっています．さて問題はこの表ですが，作り方が2種類考えられます．実行時にアセンブラで書かれたプログラムでRAMの中に表を作る方法と，ソース・プログラムの中にあらかじめ計算した表を書いてしまう方法です．前者は作表のためのプログラムがさらに増し，後者はどのようにして計算するかが問題です．後者の場合，アセンブラでは**60000**というデータは書けても**70000**というデータは書けません．M80はじめ多くの8ビットCPU用アセンブラは16ビットの数値しか扱えません．そこで65535より大きい数値は上下のワードに分けて，ふたつの数字で書かなければなりません.

この作業は電卓でも不可能ではありませんが，352ワードのデータを作るにはかなりの忍耐力が必要でしょう．そして，計算後それをエディタでソース・ファイルに入れて打鍵ミスなしで通すのは不可能です．そこで，この表を高級言語Pascal/MT＋を使ってファイルに作り出すプログラムを作りました(図6.6).

Pascal/MT＋では整数は16ビットまでですが，**BCDREAL**として10進の14桁固定小数点演算ができます．これを使用すれば変換表の計算はストレートにでき，ファイルに出力するフォーマットだけに注意すればよいことになります．表の先頭にラベルとコロンを付け，各行を**DW** ××,××…という形式で出力します．このファイルに**TABLEBCD.LIB**と名付けておけばエディタEDから簡単に取り込めます.

〈図6.5〉高速BCD↔BIN変換を行うマクロ

(a) ソース・リスト

```
1:           .Z80                    6:              P2
2:  HCONB    MACRO    P1,P2          7:              AND       60
3:           LOCAL    EM             8:              JR        Z,EM
4:           LD       A,(DE)         9:              LD        C,A
5:           P2                     10:              LD        B,0
```

```
11:             LD      IY,TABLEB+P1
12:             ADD     IY,BC
13:             LD      C,(IY)
14:             LD      B,(IY+1)
15:             ADD     IX,BC
16:             LD      C,(IY+2)
17:             LD      B,(IY+3)
18:             ADC     HL,BC
19:     EM:
20:             ENDM
21:
22:     HCOND   MACRO   P1,P2
23:             LOCAL   EM
24:             LD      A,(DE)
25:             P2
26:             P2
27:             AND     60
28:             JR      Z,EM
29:             PUSH    BC
30:             LD      C,A
31:             LD      B,0
32:             LD      IY,TABLED+P1
33:             ADD     IY,BC
34:             POP     BC
35:             LD      A,C
36:             ADD     A,(IY)
37:             DAA
38:             LD      C,A
39:             LD      A,B
40:             ADC     A,(IY+1)
41:             DAA
42:             LD      B,A
43:             LD      A,L
44:             ADC     A,(IY+2)
45:             DAA
46:             LD      L,A
47:             LD      A,H
48:             ADC     A,(IY+3)
49:             DAA
50:             LD      H,A
51:     EM:
52:             ENDM
53:     HSCNVD  MACRO   SRC,DST
54:             LOCAL   EM
55:             LD      DE,SRC
56:             CALL    HSCND
57:             IFNB    <DST>
58:             LD      IX,DST
59:             LD      (IX),C
60:             LD      (IX+1),B
61:             LD      (IX+2),L
62:             LD      (IX+3),H
63:             ENDIF
64:             JP      EM
65:     HSCND:  LD      HL,0
66:             LD      B,0
67:             LD      A,(DE)
68:             AND     15
69:             OR      A
70:             DAA
71:             LD      C,A
72:             HCOND   56,RRA
73:             INC     DE
74:             HCOND   116,RLA
75:             HCOND   176,RRA
76:             INC     DE
77:             HCOND   236,RLA
78:             HCOND   296,RRA
79:             INC     DE
80:             HCOND   356,RLA
81:             RET
82:     EM:
83:     HSCNVD  MACRO   SRC,DST
84:             LD      DE,SRC
85:             CALL    HSCND
86:             IFNB    <DST>
87:             LD      IX,DST
88:             LD      (IX),C
89:             LD      (IX+1),B
90:             LD      (IX+2),L
91:             LD      (IX+3),H
92:             ENDIF
93:             ENDM
94:             ENDM
95:     HSCNVB  MACRO   SRC,DST
96:             LOCAL   EM
97:             LD      DE,SRC
98:             CALL    HSCNB
99:             IFNB    <DST>
100:            LD      IX,DST
101:            LD      (IX),C
102:            LD      (IX+1),B
103:            LD      (IX+2),L
104:            LD      (IX+3),H
105:            ENDIF
106:            JP      EM
107:    HSCNB:  LD      HL,0
108:            LD      IX,0
109:            LD      A,(DE)
110:            AND     15
111:            LD      C,A
112:            LD      B,0
113:            ADD     IX,BC
114:            HCONB   32,RRA
115:            INC     DE
116:            HCONB   68,RLA
117:            HCONB   104,RRA
118:            INC     DE
119:            HCONB   140,RLA
120:            HCONB   176,RRA
121:            INC     DE
122:            HCONB   212,RLA
```

```
123:           HCONB    248,RRA
124:           PUSH     IX
125:           POP      BC
126:           RET
127:  EM:
128:  HSCNVB   MACRO    SRC,DST
129:           LD       DE,SRC
130:           CALL     HSCNB
131:           IFNB     <DST>
132:           LD       IX,DST
133:           LD       (IX),C
134:           LD       (IX+1),B
135:           LD       (IX+2),L
136:           LD       (IX+3),H
137:           ENDIF
138:           ENDM
139:           ENDM
140:
141:           JP       STT
142:  N1:      DW       9999H,9999H
143:  N2:      DW       -1,07FFH
144:  N3:      DS       4
145:  N4:      DS       4
146:
147:  STT:     HSCNVB   N1,N3      ┐
148:           HSCNVD   N2,N4      ├ メイン・プログラム
149:           JP       STT        ┘
150:
151:  TABLEB:                        これより204行まではTABLEBCDの数
152:    DW 1,0,2,0,3,0,4,0           (エディタにより結合)
153:    DW 5,0,6,0,7,0,8,0
154:    DW 9,0
155:    DW 10,0,20,0,30,0,40,0
156:    DW 50,0,60,0,70,0,80,0
157:    DW 90,0
158:    DW 100,0,200,0,300,0,400,0
159:    DW 500,0,600,0,700,0,800,0
160:    DW 900,0
161:    DW 1000,0,2000,0,3000,0,4000,0
162:    DW 5000,0,6000,0,7000,0,8000,0      40000がこのように出力されるが
163:    DW 9000,0                           実害はない
164:    DW 10000,0,20000,0,30000,0,-25536,0
165:    DW -15536,0,-5536,0,4464,1,14464,1
166:    DW 24464,1
167:    DW -31072,1,3392,3,-27680,4,6784,6
168:    DW -24288,7,10176,9,-20896,10,13568,12
169:    DW -17504,13
170:    DW 16960,15,-31616,30,-14656,45,2304,61
171:    DW 19264,76,-29312,91,-12352,106,4608,122
172:    DW 21568,137
173:    DW -27008,152,11520,305,-15488,457,23040,610
174:    DW -3968,762,-30976,915,7552,1068,-19456,1220
175:    DW 19072,1373
176:  TABLED:
```

```
177:     DW 1H,0H,2H,0H,3H,0H,4H,0H ←BCDコードにはHを付加する
178:     DW 5H,0H,6H,0H,7H,0H,8H,0H  （0～9には不要だが……）
179:     DW 9H,0H,10H,0H,11H,0H,12H,0H
180:     DW 13H,0H,14H,0H,15H,0H
180:     DW 13H,0H,14H,0H,15H,0H
181:     DW 16H,0H,32H,0H,48H,0H,64H,0H
182:     DW 80H,0H,96H,0H,112H,0H,128H,0H
183:     DW 144H,0H,160H,0H,176H,0H,192H,0H
184:     DW 208H,0H,224H,0H,240H,0H
185:     DW 256H,0H,512H,0H,768H,0H,1024H,0H
186:     DW 1280H,0H,1536H,0H,1792H,0H,2048H,0H
187:     DW 2304H,0H,2560H,0H,2816H,0H,3072H,0H
188:     DW 3328H,0H,3584H,0H,3840H,0H
189:     DW 4096H,0H,8192H,0H,2288H,1H,6384H,1H
190:     DW 480H,2H,4576H,2H,8672H,2H,2768H,3H
191:     DW 6864H,3H,960H,4H,5056H,4H,9152H,4H
192:     DW 3248H,5H,7344H,5H,1440H,6H
193:     DW 5536H,6H,1072H,13H,6608H,19H,2144H,26H
194:     DW 7680H,32H,3216H,39H,8752H,45H,4288H,52H
195:     DW 9824H,58H,5360H,65H,896H,72H,6432H,78H
196:     DW 1968H,85H,7504H,91H,3040H,98H
197:     DW 8576H,104H,7152H,209H,5728H,314H,4304H,419H
198:     DW 2880H,524H,1456H,629H,32H,734H,8608H,838H
199:     DW 7184H,943H,5760H,1048H,4336H,1153H,2912H,1258H
200:     DW 1488H,1363H,64H,1468H,8640H,1572H
201:     DW 7216H,1677H,4432H,3355H,1648H,5033H,8864H,6710H
202:     DW 6080H,8388H,3296H,10066H,512H,11744H,7728H,13421H
203:     DW 4944H,15099H,2160H,16777H,9376H,18454H,6592H,20132H
204:     DW 3808H,21810H,1024H,23488H,8240H,25165H
205:             END        STT
```

(b) アセンブル・リスト

```
                                            .Z80
                                    HCONB   MACRO   P1,P2
                                            LOCAL   EM
                                            LD      A,(DE)
                                            P2
                                          (途中省略)
                                            LD      (IX+2),L
                                            LD      (IX+3),H
                                            ENDIF
                                            ENDM
                                            ENDM

    0000'   C3 0013'                        JP      STT
    0003'   9999 9999               N1:     DW      9999H,9999H
    0007'   FFFF 07FF               N2:     DW      -1,07FFH
    000B'                           N3:     DS      4
    000F'                           N4:     DS      4

    0013'                           STT:    HSCNVB  N1,N3
    0013'   11 0003'        +               LD      DE,N1
    0016'   CD 002C'        +               CALL    HSCNB
    0019'   DD 21 000B'     +               LD      IX,N3
```

```
001D'   DD 71 00        +           LD      (IX),C
0020'   DD 70 01        +           LD      (IX+1),B
0023'   DD 75 02        +           LD      (IX+2),L
0026'   DD 74 03        +           LD      (IX+3),H
0029'   C3 0122'        +           JP      ..0000
002C'   21 0000         +   HSCNB:  LD      HL,0
002F'   DD 21 0000      +           LD      IX,0
0033'   1A              +           LD      A,(DE)
0034'   E6 0F           +           AND     15
0036'   4F              +           LD      C,A
0037'   06 00           +           LD      B,0
0039'   DD 09           +           ADD     IX,BC
003B'   1A              +           LD      A,(DE)
003C'   1F              +           RRA
003D'   1F              +           RRA
003E'   E6 3C           +           AND     60
0040'   28 19           +           JR      Z,..0001
0042'   4F              +           LD      C,A
0043'   06 00           +           LD      B,0
0045'   FD 21 0269'     +           LD      IY,TABLEB+32
0049'   FD 09           +           ADD     IY,BC
004B'   FD 4E 00        +           LD      C,(IY)
004E'   FD 46 01        +           LD      B,(IY+1)
0051'   DD 09           +           ADD     IX,BC
0053'   FD 4E 02        +           LD      C,(IY+2)
0056'   FD 46 03        +           LD      B,(IY+3)
0059'   ED 4A           +           ADC     HL,BC
005B'                   +   ..0001:
005B'   13              +           INC     DE
005C'   1A              +           LD      A,(DE)
005D'   17              +           RLA
005E'   17              +           RLA
005F'   E6 3C           +           AND     60
0061'   28 19           +           JR      Z,..0002
0063'   4F              +           LD      C,A
0064'   06 00           +           LD      B,0
0066'   FD 21 028D'     +           LD      IY,TABLEB+68
006A'   FD 09           +           ADD     IY,BC
006C'   FD 4E 00        +           LD      C,(IY)
006F'   FD 46 01        +           LD      B,(IY+1)
0072'   DD 09           +           ADD     IX,BC
0074'   FD 4E 02        +           LD      C,(IY+2)
0077'   FD 46 03        +           LD      B,(IY+3)
007A'   ED 4A           +           ADC     HL,BC
007C'                   +   ..0002:
007C'   1A              +           LD      A,(DE)
007D'   1F              +           RRA
007E'   1F              +           RRA
007F'   E6 3C           +           AND     60
0081'   28 19           +           JR      Z,..0003
0083'   4F              +           LD      C,A
0084'   06 00           +           LD      B,0
0086'   FD 21 02B1'     +           LD      IY,TABLEB+104
008A'   FD 09           +           ADD     IY,BC
```

```
008C'   FD 4E 00        +               LD      C,(IY)
008F'   FD 46 01        +               LD      B,(IY+1)
0092'   DD 09           +               ADD     IX,BC
0094'   FD 4E 02        +               LD      C,(IY+2)
0097'   FD 46 03        +               LD      B,(IY+3)
009A'   ED 4A           +               ADC     HL,BC
009C'                   +       ..0003:
009C'   13              +               INC     DE
009D'   1A              +               LD      A,(DE)
009E'   17              +               RLA
009F'   17              +               RLA
00A0'   E6 3C           +               AND     60
00A2'   28 19           +               JR      Z,..0004
00A4'   4F              +               LD      C,A
00A5'   06 00           +               LD      B,0
00A7'   FD 21 02D5'     +               LD      IY,TABLEB+140
00AB'   FD 09           +               ADD     IY,BC
00AD'   FD 4E 00        +               LD      C,(IY)
00B0'   FD 46 01        +               LD      B,(IY+1)
00B3'   DD 09           +               ADD     IX,BC
00B5'   FD 4E 02        +               LD      C,(IY+2)
00B8'   FD 46 03        +               LD      B,(IY+3)
00BB'   ED 4A           +               ADC     HL,BC
00BD'                   +       ..0004:
00BD'   1A              +               LD      A,(DE)
00BE'   1F              +               RRA
00BF'   1F              +               RRA
00C0'   E6 3C           +               AND     60
00C2'   28 19           +               JR      Z,..0005
00C4'   4F              +               LD      C,A
00C5'   06 00           +               LD      B,0
00C7'   FD 21 02F9'     +               LD      IY,TABLEB+176
00CB'   FD 09           +               ADD     IY,BC
00CD'   FD 4E 00        +               LD      C,(IY)
00D0'   FD 46 01        +               LD      B,(IY+1)
00D3'   DD 09           +               ADD     IX,BC
00D5'   FD 4E 02        +               LD      C,(IY+2)
00D8'   FD 46 03        +               LD      B,(IY+3)
00DB'   ED 4A           +               ADC     HL,BC
00DD'                   +       ..0005:
00DD'   13              +               INC     DE
00DE'   1A              +               LD      A,(DE)
00DF'   17              +               RLA
00E0'   17              +               RLA
00E1'   E6 3C           +               AND     60
00E3'   28 19           +               JR      Z,..0006
00E5'   4F              +               LD      C,A
00E6'   06 00           +               LD      B,0
00E8'   FD 21 031D'     +               LD      IY,TABLEB+212
00EC'   FD 09           +               ADD     IY,BC
00EE'   FD 4E 00        +               LD      C,(IY)
00F1'   FD 46 01        +               LD      B,(IY+1)
00F4'   DD 09           +               ADD     IX,BC
00F6'   FD 4E 02        +               LD      C,(IY+2)
00F9'   FD 46 03        +               LD      B,(IY+3)
00FC'   ED 4A           +               ADC     HL,BC
```

```
00FE'                   +       ..0006:
00FE'   1A              +               LD      A,(DE)
00FF'   1F              +               RRA
0100'   1F              +               RRA
0101'   E6 3C           +               AND     60
0103'   28 19           +               JR      Z,..0007
0105'   4F              +               LD      C,A
0106'   06 00           +               LD      B,0
0108'   FD 21 0341'     +               LD      IY,TABLEB+248
010C'   FD 09           +               ADD     IY,BC
010E'   FD 4E 00        +               LD      C,(IY)
0111'   FD 46 01        +               LD      B,(IY+1)
0114'   DD 09           +               ADD     IX,BC
0116'   FD 4E 02        +               LD      C,(IY+2)
0119'   FD 46 03        +               LD      B,(IY+3)
011C'   ED 4A           +               ADC     HL,BC
011E'                   +       ..0007:
011E'   DD E5           +               PUSH    IX
0120'   C1              +               POP     BC
0121'   C9              +               RET
0122'                   +       ..0000:
                                        ENDM
                                        HSCNVD  N2,N4
0122'   11 0007'        +               LD      DE,N2
0125'   CD 013B'        +               CALL    HSCND
0128'   DD 21 000F'     +               LD      IX,N4
012C'   DD 71 00        +               LD      (IX),C
012F'   DD 70 01        +               LD      (IX+1),B
0132'   DD 75 02        +               LD      (IX+2),L
0135'   DD 74 03        +               LD      (IX+3),H
0138'   C3 0246'        +               JP      ..0008
013B'   21 0000         +       HSCND:  LD      HL,0
013E'   06 00           +               LD      B,0
0140'   1A              +               LD      A,(DE)
0141'   E6 0F           +               AND     15
0143'   B7              +               OR      A
0144'   27              +               DAA
0145'   4F              +               LD      C,A
0146'   1A              +               LD      A,(DE)
0147'   1F              +               RRA
0148'   1F              +               RRA
0149'   E6 3C           +               AND     60
014B'   28 23           +               JR      Z,..0009
014D'   C5              +               PUSH    BC
014E'   4F              +               LD      C,A
014F'   06 00           +               LD      B,0
0151'   FD 21 03A1'     +               LD      IY,TABLED+56
0155'   FD 09           +               ADD     IY,BC
0157'   C1              +               POP     BC
0158'   79              +               LD      A,C
0159'   FD 86 00        +               ADD     A,(IY)
015C'   27              +               DAA
015D'   4F              +               LD      C,A
015E'   78              +               LD      A,B
015F'   FD 8E 01        +               ADC     A,(IY+1)
0162'   27              +               DAA
```

```
0163'  47           +          LD    B,A
0164'  7D           +          LD    A,L
0165'  FD 8E 02     +          ADC   A,(IY+2)
0168'  27           +          DAA
0169'  6F           +          LD    L,A
016A'  7C           +          LD    A,H
016B'  FD 8E 03     +          ADC   A,(IY+3)
016E'  27           +          DAA
016F'  67           +          LD    H,A
0170'               +  ..0009:
0170'  13           +          INC   DE
0171'  1A           +          LD    A,(DE)
0172'  17           +          RLA
0173'  17           +          RLA
0174'  E6 3C        +          AND   60
0176'  28 23        +          JR    Z,..000A
0178'  C5           +          PUSH  BC
0179'  4F           +          LD    C,A
017A'  06 00        +          LD    B,0
017C'  FD 21 03DD'  +          LD    IY,TABLED+116
0180'  FD 09        +          ADD   IY,BC
0182'  C1           +          POP   BC
0183'  79           +          LD    A,C
0184'  FD 86 00     +          ADD   A,(IY)
0187'  27           +          DAA
0188'  4F           +          LD    C,A
0189'  78           +          LD    A,B
018A'  FD 8E 01     +          ADC   A,(IY+1)
018D'  27           +          DAA
018E'  47           +          LD    B,A
018F'  7D           +          LD    A,L
0190'  FD 8E 02     +          ADC   A,(IY+2)
0193'  27           +          DAA
0194'  6F           +          LD    L,A
0195'  7C           +          LD    A,H
0196'  FD 8E 03     +          ADC   A,(IY+3)
0199'  27           +          DAA
019A'  67           +          LD    H,A
019B'               +  ..000A:
019B'  1A           +          LD    A,(DE)
019C'  1F           +          RRA
019D'  1F           +          RRA
019E'  E6 3C        +          AND   60
01A0'  28 23        +          JR    Z,..000B
01A2'  C5           +          PUSH  BC
01A3'  4F           +          LD    C,A
01A4'  06 00        +          LD    B,0
01A6'  FD 21 0419'  +          LD    IY,TABLED+176
01AA'  FD 09        +          ADD   IY,BC
01AC'  C1           +          POP   BC
01AD'  79           +          LD    A,C
01AE'  FD 86 00     +          ADD   A,(IY)
01B1'  27           +          DAA
01B2'  4F           +          LD    C,A
01B3'  78           +          LD    A,B
01B4'  FD 8E 01     +          ADC   A,(IY+1)
```

```
01B7'   27           +            DAA
01B8'   47           +            LD      B,A
01B9'   7D           +            LD      A,L
01BA'   FD 8E 02     +            ADC     A,(IY+2)
01BD'   27           +            DAA
01BE'   6F           +            LD      L,A
01BF'   7C           +            LD      A,H
01C0'   FD 8E 03     +            ADC     A,(IY+3)
01C3'   27           +            DAA
01C4'   67           +            LD      H,A
01C5'                +      ..000B:
01C5'   13           +            INC     DE
01C6'   1A           +            LD      A,(DE)
01C7'   17           +            RLA
01C8'   17           +            RLA
01C9'   E6 3C        +            AND     60
01CB'   28 23        +            JR      Z,..000C
01CD'   C5           +            PUSH    BC
01CE'   4F           +            LD      C,A
01CF'   06 00        +            LD      B,0
01D1'   FD 21 0455'  +            LD      IY,TABLED+236
01D5'   FD 09        +            ADD     IY,BC
01D7'   C1           +            POP     BC
01D8'   79           +            LD      A,C
01D9'   FD 86 00     +            ADD     A,(IY)
01DC'   27           +            DAA
01DD'   4F           +            LD      C,A
01DE'   78           +            LD      A,B
01DF'   FD 8E 01     +            ADC     A,(IY+1)
01E2'   27           +            DAA
01E3'   47           +            LD      B,A
01E4'   7D           +            LD      A,L
01E5'   FD 8E 02     +            ADC     A,(IY+2)
01E8'   27           +            DAA
01E9'   6F           +            LD      L,A
01EA'   7C           +            LD      A,H
01EB'   FD 8E 03     +            ADC     A,(IY+3)
01EE'   27           +            DAA
01EF'   67           +            LD      H,A
01F0'                +      ..000C:
01F0'   1A           +            LD      A,(DE)
01F1'   1F           +            RRA
01F2'   1F           +            RRA
01F3'   E6 3C        +            AND     60
01F5'   28 23        +            JR      Z,..000D
01F7'   C5           +            PUSH    BC
01F8'   4F           +            LD      C,A
01F9'   06 00        +            LD      B,0
01FB'   FD 21 0491'  +            LD      IY,TABLED+296
01FF'   FD 09        +            ADD     IY,BC
0201'   C1           +            POP     BC
0202'   79           +            LD      A,C
0203'   FD 86 00     +            ADD     A,(IY)
0206'   27           +            DAA
0207'   4F           +            LD      C,A
0208'   78           +            LD      A,B
```

```
0209'   FD 8E 01      +              ADC     A,(IY+1)
020C'   27            +              DAA
020D'   47            +              LD      B,A
020E'   7D            +              LD      A,L
020F'   FD 8E 02      +              ADC     A,(IY+2)
0212'   27            +              DAA
0213'   6F            +              LD      L,A
0214'   7C            +              LD      A,H
0215'   FD 8E 03      +              ADC     A,(IY+3)
0218'   27            +              DAA
0219'   67            +              LD      H,A
021A'                 +      ..000D:
021A'   13            +              INC     DE
021B'   1A            +              LD      A,(DE)
021C'   17            +              RLA
021D'   17            +              RLA
021E'   E6 3C         +              AND     60
0220'   28 23         +              JR      Z,..000E
0222'   C5            +              PUSH    BC
0223'   4F            +              LD      C,A
0224'   06 00         +              LD      B,0
0226'   FD 21 04CD'   +              LD      IY,TABLED+356
022A'   FD 09         +              ADD     IY,BC
022C'   C1            +              POP     BC
022D'   79            +              LD      A,C
022E'   FD 86 00      +              ADD     A,(IY)
0231'   27            +              DAA
0232'   4F            +              LD      C,A
0233'   78            +              LD      A,B
0234'   FD 8E 01      +              ADC     A,(IY+1)
0237'   27            +              DAA
0238'   47            +              LD      B,A
0239'   7D            +              LD      A,L
023A'   FD 8E 02      +              ADC     A,(IY+2)
023D'   27            +              DAA
023E'   6F            +              LD      L,A
023F'   7C            +              LD      A,H
0240'   FD 8E 03      +              ADC     A,(IY+3)
0243'   27            +              DAA
0244'   67            +              LD      H,A
0245'                 +      ..000E:
0245'   C9            +              RET
0246'                 +      ..0008:
                                     ENDM
0246'   C3 0013'                     JP      STT

0249'                        TABLEB:
0249'   0001 0000             DW 1,0,2,0,3,0,4,0
024D'   0002 0000
0251'   0003 0000
0255'   0004 0000
0259'   0005 0000             DW 5,0,6,0,7,0,8,0
025D'   0006 0000
                             (途中省略)
035D'   1D80 042C
0361'   B400 04C4
0365'   4A80 055D             DW 19072,1373
```

```
0369'                     TABLED:
0369'   0001 0000          DW 1H,0H,2H,0H,3H,0H,4H,0H
036D'   0002 0000
0371'   0003 0000
0375'   0004 0000
0379'   0005 0000          DW 5H,0H,6H,0H,7H,0H,8H,0H
037D'   0006 0000
                          (途中省略)
04FD'   6592 0132
0501'   3808 1810          DW 3808H,21810H,1024H,23488H,8240H,25165H
0505'   1024 3488
0509'   8240 5165
                                   END      STT
```

〈図6.6〉 BCD↔バイナリ変換表作成プログラム

```
 1:
 2:  PROGRAM TABLEBCD;
 3:  TYPE   H = STRING[1];
 4:  VAR    F : TEXT;
 5:       K,N: REAL;
 6:       RSLT: INTEGER;
 7:
 8:  PROCEDURE CAL(I:INTEGER;D:REAL;C:H);
 9:  BEGIN
10:    N := N+K;
11:    IF I MOD 4 = 1 THEN
12:      BEGIN
13:        WRITELN(F);
14:        WRITE(F,' DW ')
15:      END
16:    ELSE
17:      WRITE(F,',');
18:    WRITE(F,TRUNC(N-TRUNC(N/D)*D),C,',',TRUNC(N/D),C)
19:  END;
20:
21:  PROCEDURE TABLE (A,B:INTEGER;D:REAL;C:H);
22:  VAR I,J: INTEGER;
23:  BEGIN
24:    K := 1 ;
25:    FOR J:=1 TO B DO
26:      BEGIN
27:        N := 0 ;
28:        FOR I:=1 TO A DO
29:          CAL(I,D,C);
30:        K := K*(A+1);
31:        WRITE('¥')
32:      END
33:  END;
34:  BEGIN
35:    ASSIGN (F,'TABLEBCD.LIB');
36:    REWRITE (F);
37:    WRITELN ('10ｼﾝ ﾍﾝｶﾝ ﾋｮｳ ｻｸｾｲ');
38:    WRITELN(F);
39:    WRITE (F,'TABLEB:');
40:    TABLE (9,8,65536.0,'');
```

```
41:     WRITELN(F);
42:     WRITE (F,'TABLED:');
43:     TABLE (15,7,10000,'H');
44:     WRITELN(F);
45:     CLOSE(F,RSLT) ;
46:     IF RSLT=255 THEN
47:       WRITELN('CLOSE ERROR')
48:     ELSE
49:       WRITELN('ヲワリマシタ  TABLEBCD.LIB ヲ サクセイ')
50:  END.
51:
```

6.4 アセンブラとコンパイラのリンク ──バック・グラウンド・プリンタ

CP/M での通常のプリントは **TYPE** コマンドか **PIP** による転送で行います．両者ともコマンドから抜け出ればプリントを続けることができません．バック・グラウンド・プリンタは，コマンドから抜け出てもインタラプト処理によってバッファ・エリヤ内のデータをプリンタへ出力し続けるためのプログラムです．

PC-CP/M は BASIC インタプリタ用 ROM の中の基本サブルーチンを使用しているので，プログラム内のシステム・コールや CCP へ戻った時にバンク切り替えで下半分の RAM が切り離されます．また，Z80 のモード 2 のインタラプトを使用していますが，Z80 ファミリの周辺 LSI ではなく，8214 (インタラプト・コントローラ)を使用しています．したがって，インタラプト処理はやや複雑化します．それでも I/O ポートの入出力が多いので，アセンブラで書く方が書きやすいでしょう．

ところが，プリントするデータをディスクから持ってくるのは何といってもコンパイラの方が有利です．そこで，プリント・アウトの動作とそのためのイニシャライズをアセンブラで書き，ファイル名を指定したりファイルを読み取ったりする部分を Pascal/MT+ で書いて両者のオブジェクト・コードをリンクし，1本のプログラムとして使用することにしました．

6.4.1 イニシャライズとプリント・アウト

アセンブラで書く部分は，イニシャライズとプリント・アウト・ルーチンです．イニシャライズはタイマ IC(8253 を使用，5.2 節参照)とインタラプト・モード，ベクタ・ページ(I レジスタ)セットなどのハードウェア的な部分と，バッファ・メモリ・エリヤをクリア (**0**

でなく **1AH** にクリヤ)，ベクタ・アドレス・セット，そして，インタラプト処理ルーチンをどの状態でインタラプトが入っても正しく受けられるアドレスへ移すというソフトウェア的な部分によって作られています(図6.7).

インタラプト処理ルーチンは，一度 TPA のどこか(リンクしなければ定まらない)のエリアにロードされた後，転送プログラムによって **0FF40H** からのエリアに移され，その後インタラプトによって起動されます．このエリアはCP/Mもその他の一般のプログラムもアクセスすることはありません．

実行時に別の番地に移して正しいアドレスを示す機能は4.6節で説明したものです．

インタラプトの前後処理が多いのは，バンク切り替えとデイジィ・チェーンでない優先順位処理のためです．Z80専用LSIを使用していれば，**PUSH** と **POP** だけで済みます．

プリント・ルーチンはプリンタがBUSYかポインタが指しているメモリ内容が **1AH** のときはすぐリターンし，そうでないときは，BUSYになるか，**1AH** を検出するまでプリンタに出力してポインタを進めます．ポインタを進めてバッファ・エリアを越えたら，バッファ・エリアの先頭にもどして動作を続けます．最初に起動した時は，バッファ内容がすべて **1AH** にクリアされていますからインタラプトの実質的な処理は行われません．

〈図6.7〉 Pascal/MT＋ とリンクされるインタラプト処理ルーチン

```
                                    .Z80
                            ; ｲﾝﾀﾗﾌﾟﾄ ｼｮﾘ ﾓｼﾞｭｰﾙ
                            ; ••••••••••••••••••••••••
                            ;
                            ; ｲﾆｼｬﾗｲｽﾞ ｲﾝﾀﾗﾌﾟﾄ etc.
                            ;
                                    PUBLIC  INITM
                            ;
0000'   3A 0034             INITM:  LD      A,(34H)   ;ﾀｲﾏ ﾓｰﾄﾞ 2
0003'   D3 EB                       OUT     (0EBH),A ;8253
0005'   C1                          POP     BC
0006'   E1                          POP     HL          ;Get Time Const from STACK
0007'   C5                          PUSH    BC        ;RET ADS on STACK
0008'   7D                          LD      A,L
0009'   D3 E8                       OUT     (0E8H),A
000B'   7C                          LD      A,H
000C'   D3 E8                       OUT     (0E8H),A        ;ﾀｲﾏ ｲﾝﾀﾊﾞﾙ ｾｯﾄ
000E'   3E FF                       LD      A,-1
0010'   32 EA55                     LD      (0EA55H),A
0013'   D3 E4                       OUT     (0E4H),A
0015'   3E EC                       LD      A,0ECH
0017'   ED 47                       LD      I,A           ;ﾍﾞｸﾀ HI ads ｾｯﾄ
0019'   ED 5E                       IM      2             ;ﾓｰﾄﾞ 2 ﾜﾘｺﾐ
001B'   21 FF40                     LD      HL,TMVECT  ;ﾍﾞｸﾀ ﾃｰﾌﾞﾙ ｾｯﾄ
001E'   22 EC06                     LD      (0EC06H),HL
```

```
0021'   21 FFAF                 LD      HL,SWVECT
0024'   22 EC0A                 LD      (0EC0AH),HL
0027'   3E 90                   LD      A,90H   ;8255ｲﾆ. ﾀｲﾏ ｸﾛｯｸ Enable
0029'   D3 EF                   OUT     (0EFH),A
002B'   FB                      EI
002C'   C9                      RET
                        ;
                        ; TPAｶﾗ ｱﾝｾﾞﾝ ﾅ ｴﾘｱ ﾆ ｲﾝﾀﾗﾌﾟﾄ
                        ;  ｼｮﾘ ﾌﾞ ｦ ｳﾂｽ
                        ;
                                PUBLIC  TRANS
                        ;
002D'   F3              TRANS:  DI
002E'   11 FF40                 LD      DE,TMVECT  ;ﾜﾘｺﾐ ﾙｰﾁﾝ ﾃﾝｿｰ
0031'   21 0052'                LD      HL,SRCVT
0034'   01 007B                 LD      BC,SWVEE-TMVECT+1
0037'   ED B0                   LDIR
0039'   C1                      POP     BC        ;RET ADS
003A'   E1                      POP     HL        ;BUFTOP:INTEGER
003B'   D1                      POP     DE        ;BUFEND:INTEGER
003C'   22 FFAB                 LD      (BUFTOP),HL
003F'   22 FFA9                 LD      (POINT),HL
0042'   ED 53 FFAD              LD      (BUFEND),DE
0046'   C5                      PUSH    BC        ;RET ADS on STACK
0047'   3E 1A                   LD      A,1AH      ;ﾊﾞｯﾌｧ ｸﾘｱ EOF code
0049'   A7              FLOP:   AND     A
004A'   77                      LD      (HL),A
004B'   ED 52                   SBC     HL,DE
004D'   ED 5A                   ADC     HL,DE
004F'   38 F8                   JR      C,FLOP
0051'   C9                      RET
                        ;
                        ;
0052'                   SRCVT:                    ;TPA ﾆ ﾛｰﾄﾞ ｻﾚﾙ ｱﾄﾞﾚｽ
                        ;
                        ; ﾀｲﾏ ｲﾝﾀﾗﾌﾟﾄ ﾆ ﾖｯﾃ ﾊﾞｯﾌｧ ﾆ ﾊｲｯﾃｲﾙ ﾓｼﾞｺｰﾄﾞ ｦ
                        ;   ﾌﾟﾘﾝﾀ ﾆ ｵｸﾘﾀﾞｽ
                        ;
                                .PHASE  0FF40H  ;ｱﾝｾﾞﾝ ｴﾘｱ
FF40    F5              TMVECT: PUSH    AF
FF41    3E 08                   LD      A,8       ;ﾓﾆﾀ ﾗﾝﾌﾟ ON
FF43    D3 ED                   OUT     (0EDH),A ;ﾓﾆﾀ ﾎﾟｰﾄ
FF45    E5                      PUSH    HL
FF46    D5                      PUSH    DE
FF47    2A FFA9                 LD      HL,(POINT)
FF4A    ED 5B FFAD              LD      DE,(BUFEND)
FF4E    3A EA55                 LD      A,(0EA55H)
FF51    F5                      PUSH    AF            ;ﾜﾘｺﾐ ﾚﾍﾞﾙ ﾀｲﾋ
FF52    3A E9FF                 LD      A,(0E9FFH)
FF55    F5                      PUSH    AF            ;ﾊﾞﾝｸ ﾀｲﾋ
FF56    3E 03                   LD      A,3        ;ﾚﾍﾞﾙ 3 ﾆｼﾃｻｲﾜﾘｺﾐ ｷﾝｼ
FF58    32 EA55                 LD      (0EA55H),A
FF5B    D3 E4                   OUT     (0E4H),A  ;ﾚﾍﾞﾙ ｾﾝﾀｸ ﾎﾟｰﾄ
FF5D    3E 11                   LD      A,11H         ;ﾊﾞﾝｸ 0 RAM ｾﾝﾀｸ
FF5F    32 E9FF                 LD      (0E9FFH),A
FF62    D3 E2                   OUT     (0E2H),A  ;ﾊﾞﾝｸ ｾﾝﾀｸ ﾎﾟｰﾄ
```

```
FF64    FB                    EI
FF65    DB 40         LPLOP:  IN      A,(40H)   ;ﾌﾟﾘﾝﾀ BUSY?
FF67    CB 47                 BIT     0,A
FF69    20 23                 JR      NZ,RETINT
FF6B    7E                    LD      A,(HL)
FF6C    FE 1A                 CP      1AH
FF6E    28 1E                 JR      Z,RETINT
FF70    D3 10                 OUT     (10H),A
FF72    36 1A                 LD      (HL),1AH
FF74    3A EA67               LD      A,(0EA67H)
FF77    CB 87                 RES     0,A
FF79    D3 40                 OUT     (40H),A
FF7B    CB C7                 SET     0,A
FF7D    D3 40                 OUT     (40H),A
FF7F    32 EA67               LD      (0EA67H),A
FF82    A7                    AND     A
FF83    ED 52                 SBC     HL,DE
FF85    ED 5A                 ADC     HL,DE
FF87    38 DC                 JR      C,LPLOP
FF89    2A FFAB               LD      HL,(BUFTOP)
FF8C    18 D7                 JR      LPLOP
FF8E    F3            RETINT: DI
FF8F    F1                    POP     AF        ;ﾊﾞﾝｸ ｶｲﾌｸ
FF90    32 E9FF               LD      (0E9FFH),A
FF93    0F                    RRCA
FF94    C6 10                 ADD     A,10H
FF96    D3 E2                 OUT     (0E2H),A
FF98    F1                    POP     AF        ;ﾏｽｸ ﾚﾍﾞﾙ ｶｲﾌｸ
FF99    32 EA55               LD      (0EA55H),A
FF9C    D3 E4                 OUT     (0E4H),A
FF9E    22 FFA9               LD      (POINT),HL
FFA1    D1                    POP     DE
FFA2    E1                    POP     HL
FFA3    AF                    XOR     A         ;ﾓﾆﾀ ﾗﾝﾌﾟ OFF
FFA4    D3 ED                 OUT     (0EDH),A
FFA6    F1                    POP     AF
FFA7    FB                    EI
FFA8    C9                    RET
FFA9                  POINT:  DS      2
FFAB                  BUFTOP: DS      2
FFAD                  BUFEND: DS      2
                      ;
                      ; ｽｲｯﾁ ﾆ ﾖｯﾃ ﾀｲﾏ ﾜﾘｺﾐ ｦ ﾄﾒﾙ
                      ; OS/TP ｲｽﾞﾚﾆﾓ ﾕｳｺｳ
                      ;
FFAF    F5            SWVECT: PUSH    AF
FFB0    3E 34                 LD      A,34H
FFB2    D3 EB                 OUT     (0EBH),A  ;ﾓｰﾄﾞ ｾｯﾄ ﾃﾞ ﾀｲﾏ ｦ ﾄﾒﾙ
FFB4    3E FF                 LD      A,-1
FFB6    D3 E4                 OUT     (0E4H),A
FFB8    F1                    POP     AF
FFB9    FB                    EI
FFBA    C9            SWVEE:  RET
                              .DEPHASE
                      ;
                              END          ;ﾓｼﾞｭｰﾙ ｵﾜﾘ
```

6.4.2 ファイル読み取りとインタラプト・ルーチンへデータを渡す

こちらはPascalで書く部分です．プログラムが起動するとすぐアセンブラにバッファ・エリアやタイマのインターバル値を知らせてイニシャライズさせます．その後，ファイル名を指定してファイルを1文字ずつ読み取り，バッファ・エリアへ書いて行きます．バッファ・エリアは16384文字のアレイとして宣言していますから，何番地かは考える必要はなく，コンパイル後にアセンブラ出力とリンクすると，この番地の値がイニシャライズ・ルーチンに与えられるようにしてあります(図6.8)．

バッファのアレイ **BUF** は **TRANS** で **1AH** にイニシャライズされています．そこへファイルから読み取った文字を書いて行きますが，**1AH** でないバッファには書き込みません．これは，プリント・ルーチンがまだ処理していないことを示していますから **1AH** になるまで待ちます．この方式でバッファ・エリアがFIFOの働きをします．ファイルを読み取ってバッファ・エリアに書くプログラムとインタラプトで起動されてバッファ内容をプリントするプログラムとの間で，FIFO動作のためのデータ量やデータ・アドレスのやり取りは行いません．

〈図6.8〉 Pascal/MT＋ で作ったプリント・ルーチン

```
 1:
 2:  PROGRAM PRNT ;
 3:  TYPE PAOC = PACKED ARRAY [1..256] OF CHAR ;
 4:       PTR = ^BYTE ;
 5:  VAR FI : FILE OF PAOC ;
 6:      CH : CHAR ;
 7:      BUFEND,BUFTOP : PTR ;
 8:      CF : STRING[13] ;
 9:      I,RSLT,TMUNIT : INTEGER  ;
10:      BUF : PACKED ARRAY [1..16384] OF CHAR ;
11:
12:      EXTERNAL PROCEDURE TRANS (BUFEND,BUFTOP : PTR) ;
13:
14:      EXTERNAL PROCEDURE INITM (TMUNIT : INTEGER) ;
15:  BEGIN
16:    BUFTOP := ADDR(BUF) ;
17:    BUFEND := ADDR(BUF[16384]) ;
18:    TRANS(BUFEND,BUFTOP) ;
19:    INITM(2000) ;(*2ﾐﾘｾｺﾝﾄﾞ*)
20:    I := 1 ;
21:    REPEAT
22:      WRITELN ;
23:      WRITE(' ﾌｧｲﾙﾒｰ ? ') ;
24:      INLINE($3E / $FF / $32 / $EA55 / $D3 / $E4) ;
25:      READ(CF) ;
26:      IF CF<>'Q' THEN
27:        BEGIN
```

```
28:          OPEN(FI,CF,RSLT);
29:          IF RSLT=255 THEN
30:            WRITELN('ｿﾉ ﾌｧｲﾙ ﾜ ﾖﾒﾏｾﾝﾃﾞｼﾀ ')
31:          ELSE
32:            BEGIN
33:              REPEAT
34:                IF BUF[I]=CHR(26) THEN
35:                  BEGIN
36:                    CH := GNB(FI) ;
37:                    BUF[I] := CH ;
38:                    I := I+1 ;
39:                    IF I > 16384 THEN
40:                      I := 1
41:                  END
42:              UNTIL (CH=CHR(26)) OR (CH=CHR(255)) ;
43:              CLOSE(FI,RSLT) ;
44:              IF RSLT = 255 THEN
45:                WRITELN('CLOSE ERROR ')
46:              ELSE
47:                WRITELN('                    ｵﾜﾘﾏｼﾀ   ')
48:            END
49:        END
50:    UNTIL CF='Q'
51:  END.
52:
```

6.4.3 Pascal からアセンブラへのパラメータ伝達

Pascal/MT＋においては，内部，外部や Pascal かアセンブラかにかかわらずメイン・ルーチンとサブルーチン(Pascal では手続きや関数)とのデータ伝達はスタックを通して行います．スタックを使用する最大の理由は，サブルーチンの自己呼び出し(再帰手続き，再帰関数)を可能にすることです．

メイン・プログラムは，サブルーチンに与えるべきデータ(実パラメータ)をスタックに積み上げてから **CALL** します．サブルーチンでは，このスタックの上から必要なデータをしかるべき数だけ取り出し，処理結果をしかるべき数だけスタックに積み上げて **RET** します．返すデータがないサブルーチンではスタックには何も積まずに **RET** します．

ここで非常に重要なことは，サブルーチン側で取り出すべきデータの量と返すべきデータの量だけは絶対に間違ってはいけないということです．サブルーチンへ入った時には，スタックの上にそれまでに保留されたデータや帰り番地が積まれていますから，これがひとつ狂えばたちまちプログラムは暴走します．データの量さえ正しければ，処理結果が正しくなくても大問題ではありません．これは処理結果を見ながら対処できます．

図6.8では12行と14行で外部手続きを宣言しています．このときのカッコ内に渡すパラメータの仮の名前と，そのデータ**TYPE**が書かれています．この**TYPE**とパラメータの数によって，スタックに積まれるデータ量が決まります．手続き**TRANS**では**PTR**タイプのデータが2つ，合計4バイトが渡されることになります．**INITM**では2バイトです．

メイン・ルーチンでは宣言順にスタックへ積んで最後に**CALL**命令が入りますから，サブルーチン側では常にスタックの上には帰り番地が乗っており，その下に最後に宣言されたパラメータが乗っています．**TRANS**では(**BUFEND，BUFTOP　：　PTR**)と宣言していますから，サブルーチンへ入った時点では帰り番地，**BUFTOP, BUFEND**の順に積まれているわけです．

図6.7では**PUBLIC　TRANS**でこの名前を外部から呼べるようにしておき，**TRANS**：の5ステップ目からスタックの処理をしています．直前の**LDIR**で**FF40H**からのアドレスに移されたインタラプト処理ルーチンにスタックから受け取ったバッファ・エリアを教えています．帰り番地は必ずしもスタックに積まなくても構いませんが，一応スタックに積んでおき，ルーチンの終わりで**RET**するのが常套手段です．

図6.7ではデータのやり取りなどの処理をわかりやすくするためにマクロは一切使用していませんが，コンパイラとのデータのやり取りはワンパターンですからマクロ化は簡単にできます．要はデータの量を間違えないことです．データのタイプによるバイト数のちがいは，コンパイラのマニュアルに書いてあります．

6.4.4 リンクの操作

コンパイラとアセンブラの出力モジュールをリンクするわけですが，その前に各々のコンパイルとアセンブルをしなければなりません．ここに取り上げた例に限らず，アセンブラとコンパイラではコンパイラ側をメイン・プログラムとして扱います．これはコンパイラの起動手順が面倒なのでアセンブラ側から飛び込んでも起動しにくいからです．

アセンブラでの処理を先に実行する必要があれば，図6.8の16〜19行のように，他の処理に先立ってアセンブル手続きを実行すればよいのです．

リンクには2通りの方法があって，ひとつはPascal/MT＋に付属のリンカLINKMTを使用する方法，もうひとつはL80を使用する方法です．後者の方法はPascal側のオブジェクトを一度フォーマット変換しなければならないので，今回は前者の方法としました．

そこでアセンブラに戻りますが，**LINKMT**に入力できるファイルは**.REL**(L80での入力標準＝M80の出力標準エクステンション)を指定しても受け付けてくれません(マニュ

〈図6.9〉コンパイルとリンクの操作結果

```
MTPLUS B:PRNT $BZ                        Pascal/MT＋を起動．そのあとはオプション指示
Pascal/MT+          Release 5.5
(c) 1981 MT MicroSYSTEMS, Inc.

BCD real format selected ……オプションB(ここでは不使用)
Enhanced Z80 object code selected……オプションZ(入れなくてもよい)
CP/M-80 version
++++++
Source lines:    102
Symbol Table Initialization
Available Memory: 16349
User Table Space: 12353
V5.5 Phase 1
##
Remaining Memory: 11859           コンパイル中に進行状況を出力
V5.5 Phase 2                      している．エラーがあればその
INBUF    9                        メッセージも出る．
JCOBUF  106
PRNT

Lines :       102
Errors:         0
Code  :       796
Data  :     16868
Pascal/MT+ 5.5 Compilation Complete

LINKMT B:PRNT,B:D2,PASLIB/S/M/D:3000 ←データ・エリヤの先頭を指定
Link/MT+ Release 5.5                 └シンボル・マップ出力

Processing file- B:PRNT     .ERL ……エクステンションは(ERL)でないと受け付けない
                                      (コマンド・ラインで指定してもうまく働かない)
Processing file- B:D2       .ERL

Processing file- PASLIB     .ERL
                 必ずリンクしなければならない
                 実行時ルーチン群
Undefined Symbols:

                                  [    ] 内はD2. ERLで定義されている
No Undefined Symbols              (    ) 内はPRNT. ERLで定義されて
                                         D2. ERLで参照される
External Symbol Map: ……オプション指示なければ出力されない

MOVERI   1F24    MOVELE   1F07    PUT      10D7    DELETE   0E33
MOVE     1F07    PURGE    0EEA    CLOSED   0905    FILLCH   1F46
GET      0FB4    RESULTI  7457    IORESU   1925    ASSIGN   0B67
RESET    0626    OPENX    058D    SYSMEM   76B5    CLOSE    0902
GNB      16F9    INPUT    71E4    OPEN     058A    OUTPUT   72A7
[INITM   041C]   [TRANS   0449]   JCOBUF   0116    INBUF    0113
BUF      31E0    I        31DE    J        31DC    RSLT     31DA
(TMUNIT  31D8)   CF       31CA    (BUFEND  31C8)   (BUFTOP  31C6)
CH       31C4    FI       3000
                 ↑
                 └これがデータ・エリヤの先頭にある
```

```
0062 (decimal) records written to .COM file
   └─約8Kバイト

Total Data: 46B7H bytes
Total Code: 1E86H bytes
Remaining : 7FC2H bytes

Link/MT+ Release 5.5 processing completed
```

アルには指定できそうに書いてありますが……).そこで,アセンブル時にオブジェクト・ファイル名を指定して**.ERL** ファイルを作成させます.

LINKMTで両者をリンクするときには,コンパイラ側がメインですから,コンパイラのオブジェクトを第一に,アセンブラのオブジェクトを第二に,そしてコンパイラに必要な実行時モジュール群(いわゆるランタイム・ライブラリ)を第三に指定します.また,オプションとしてライブラリのサーチを**/S** で指定します.

その他のオプションは全く指定なしでも動作には影響しませんが,ROMとRAMに分ける場合などの参考のためにデータ・エリアを指定する**/D**:オプションを指定し,各ラベルの配置をデバッグ時に利用するためにシンボル表を出力させます.以上の操作結果は図6.9のようになります.

このように,M80の出力オブジェクト・ファイルはエクステンションの変更だけでLINKMTに難無く入力でき,他のモジュールとリンクされます.LINKMTとL80との大きな違いは,第一にL80はリンク・ローダであり,原則として実メモリ・エリアに配置してリンク作業を行うのに対して,LINKMTでは実メモリと無関係にバイナリ・ファイル(**.COM**)または**HEXA** ファイルを作成することです.第二はL80では外部記号を6文字までしか判断できないのに対して,LINKMTでは7文字まで判断できることです.LINKMTはローダ機能がない以外は,L80よりも上回った仕様になっているといえます.もちろん,アセンブラのみのファイルもLINKMTでリンクできるのは当然です.

アセンブラとリンクして使えるのはPascalだけではありません.コンパイラのほとんどはライブラリをアセンブラで書いてあり,必然的にアセンブラとのリンクができるようになります.数あるコンパイラの中で特に筆者がPascalを選んだのは自然語の意味に最も近い表現をとっている点とデータ,プログラムとも大きな構造を扱える点です.おそらくPascalまたはこれに類似した言語がハードウェアの変化にかかわらず生き続ける息の長い言語になるでしょう.

アセンブラのみに頼る時代は遠からず去ります．マクロ・アセンブラを踏み台に，高級言語を並用できるように目標を据えてください．

参考文献

(1) ユーティリティ・ソフトウェア・マニュアル，㈱アスキー

(2) Alan R. Miller，友枝英一訳，マスタリングCP/M，CQ出版社

(3) 前田英明，マクロアセンブラの使い方，工学図書出版

(4) 牟田慎一郎，PC-Techknow 8000 mkII，㈱システムソフト

(5) Pascal/MT＋ Reference Manual, Pascal/MT＋ Programmer's Guide, SPP User's Guide, Digital Research

(6) 中野正次，ディジタル回路設計ノウハウ，CQ出版社

使用ソフトウェア

(1) ユーティリティ・ソフトウェア・パッケージ

MACRO-80 V3.44
LINK-80 V3.44
(Microsoft)

(2) Pascal/MT＋ with SPP Release 5.5
(Digital Research)

キーワード

〔ア　行〕

〔カ　行〕

〔サ 行〕

〔記号，数字〕

あとがき

人間がコンピュータより優れているのは直感力だといわれています．また，イメージ的，パターン的判断力ともいわれます．機械語のみのアセンブラでは**LD**や**CALL**命令，また**IN, BIT, JR**の組み合わせがただただ数多く並んでいるだけで，人間持ち前の直感力を発揮するのに適していません．マクロ・アセンブラのマクロ機能は，機械語列をより直感力を働かせやすい表現に近づけて，構成，推察，検査，変更などの効率を上げるための強力な手段といえます．

ところで，本当に人間は直感力において機械より優れているのでしょうか？　一般的に優れているとしても大差があるとは思えません．しかし，よく鍛練された芸術家やスポーツ選手などは，確かに機械よりはるかに勝る直感力とパターン処理力を持っているようです．

直感力といえば，以前，ディジタル回路のプリント配線パターンを設計していたとき，最後の詰めになって何本かの配線が通らなくなり，ついに全体を始めからやり直す羽目に陥ることが何度もありました(筆者に限らず…ですが)．そんな時，たまたま流行(はや)り出したのがプラパズルでした．

プラパズムというのは，原理が非常に単純で数学的なものです．その基本は，正Ｎ角形をＭ個平面上で連結して作られるすべて（Ｌ種類）の形をプラスティック板で作り，Ｍ×Ｌ目(もく)の外枠の中にすき間なく詰めるパズルです．平面を埋めるためにはＮは３，４，６の３種類しか使えません．また，面積はすべて同じで，同じ形は２つとないことになります．

原理はこれだけですから誰でも作れるわけですが，市販品では Ｎ＝４（すなわち正方形)，Ｍ＝６のもの(したがって，Ｌは必然的に35になる)が最も多くの組み替えができるとされています．筆者の推察では数十億通り以上と思われます．

プラパズルには正解というものはありません．外枠の中に入れるという制限の中で自由にパターンを作るものです．そこで話は戻って，プラパズルをいろいろと試しているうちに，プリント配線パターンの設計がより速く適確にできるようになり，以前の苦労が不思議にさえ思えてきたのです．人間の直感力やパターン処理力は訓練によって鍛えられていくものだと痛感しました．

その後，このプラパズルでいろんな模様を作りましたが，さらに可能性を追求して文字を書くことも不可能ではなさそう…と思い挑戦してみました．とはいっても，限られた図形の組み合わせですから，どんな文字でも書けるわけではありません（文字以外の部分を無視しても／)．文字パターンを作って，その残りを埋めるのは簡単ではありませんが，筆者の姓をカタカナで書くことはできました．これには１日４時間程度で約３ヵ月かかりました．そんなことに３ヵ月もかける価値があると思う人は異常です．しかし，価値がないのに敢えてやってしまうのは……もっと異常でしょうか．

最後になりましたが，ディジタルリサーチ・ジャパン㈱からはPascal/MT＋，㈱アスキーからはMacro-80の最新バージョンを確認のためにお借りしました．この場を借りてお礼を申し上げます．

著者略歴

中野　正次（なかの　まさつぐ）

1947年　福井県に生まれる

1969年　金沢大学工学部電子工学科卒業

1969年　日本電子株式会社　入社

1975年　ナスコ株式会社　入社

1982年　同社退社

現　在　技術コンサルタント

著　書「抵抗，コンデンサの使い方」（共著，CQ出版社，1980）

「ディジタル回路設計ノウハウ」（CQ出版社，1984）

実戦マクロ・アセンブラ活用法

昭和60年4月5日　初版発行

著　者　中　野　正　次

発行人　飛　坐　博

発行所　CQ出版株式会社

東京都豊島区巣鴨1－14－2　（〒170）

電話 03(947)6311(代)　振替 東京0-10665

定価 1,800円

落丁，乱丁本はお取り替えします．

写植　都写真植字社　　印刷・製本　美和印刷

ISBN4-7898-3157-4 C3055 ¥1800E